Incorporating Dwell Time in Session-Based Recommendation System With Graph Neural Networks

Chen Jia
STUDENT NUMBER: 2040771

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN DATA SCIENCE & SOCIETY
DEPARTMENT OF COGNITIVE SCIENCE & ARTIFICIAL INTELLIGENCE
SCHOOL OF HUMANITIES AND DIGITAL SCIENCES
TILBURG UNIVERSITY

Thesis committee:

Giovanni Cassani
Sharon Ong

Tilburg University
School of Humanities and Digital Sciences
Department of Cognitive Science & Artificial Intelligence
Tilburg, The Netherlands
December 2020

Preface

This thesis is one of the most important fruits of my Data Science study until now. My personal learning curve grew like a logarithmic function in the past few months. Before writing this master thesis, I never thought I could finish such a complicated experiment on my own. When I have to do the whole experiment again because of a typo in the hyperparameters, I realized how hard that scientists tried to make our life easier and more colorful.

I would like to thank my supervisor Dr. Giovanni Cassani who told me that a fancy result is not the most important thing in scientific research, the truth is. I would also thank my parents who supported me and gave me this opportunity to finish my master's degree. I thank my friend Si Yu Fang who gave a lot of supports and advice on this thesis. Finally, I am grateful to my girlfriend Ya Qi Chen who always encourages me when I am blue and always available when I need help.

I hope you enjoy this thesis.

Incorporating Dwell Time in Session-Based Recommendation System with Graph Neural Networks

Chen Jia

Recommendation system (RS) helps online customers to filter the products they want from the information flood. As a branch of RS, Session-Based Recommendation System (SBRS) uses users' anonymous sessions only to generate recommendations. Although the state-of-the-art Recurrent Neural Networks (RNN)-based SBRSs achieved promising performances, these models neglected the value of item transitions. For obtaining better item representations and taking the item transitions into account, Graph Neural Networks (GNN) is applied on SBRS and achieved better performance than RNN-based SBRS. For further improving the performance of GNN-based SBRS, this thesis explores the possibility of adding dwell times as a new feature in GNN-based SBRS. The dwell times are incorporated in edge weights of session graphs by multiplying them and occurrences of item transitions. For determining the correlation of dwell times and relationship strengths of items, both normalization and inverse-normalization are used to scale the dwell times. The results show that there is a negative correlation between dwell times and relationship strengths of items. However, the research failed to prove that adding dwell times in session graphs can improve the performance of GNN-based SBRS.

1. Introduction

With the rapid development of internet and web applications, people can easily get access to the information they want with an affordable cost. However, information overload appears and gradually became a huge problem in this cyber era (Hemp 2009). Too much information is not always a good thing, the information flood can make it very difficult for people to find the products they want from an online store (Leimstoll and Stormer 2007). Thanks to the advancements in machine learning, Recommendation System (RS) was invented by Goldberg et al. (1992) to help customers filter the products they want out of the huge amount of groceries based on users' preferences and interests. After years of applications and developments, nowadays, RS can significantly save users' time in browsing and thus improve their shopping experience (Verhoef et al. 2005). For internet companies such as Netflix and Amazon, a good RS helps them to generate more profit and stand out from their competitors by providing better user experience (Jannach and Jugovac 2019). Among all the RSs, Collaborative Filtering (CF) and Content-Based Recommendation System (CBRS) are the two most widely used systems. Both of these two systems require user's long-term histories (e.g., users' browsing history, profile and cookies) or behaviors such as ratings to make reliable recommendations. Nevertheless, the problem of conventional RSs is that most of them ignore users' current interests (Hidasi et al. 2015). For some online stores such as fast fashion and sports shops, users' current preferences are more important than their profiles and histories in making recommendations (Hidasi et al. 2015). For example,

a customer wants to buy some winter clothes but got a long list of digital gadgets only because he searched "scarf" once and "wireless mouse" four times. Another drawback of CF and CBRS is the high cost of users' information. Nowadays, governments and public administrations start to aware the importance of information security (Scholl 2018; Topping 2017). Many legal safeguards such as General Data Protection Regulation (GDPR) were set to ensure internet users' information will not be used without their permissions (Li, Yu, and He 2019). These limitations have led to the further increasing cost of time and money regarding users' information retrieval (i.e., the process of obtaining user information such as demographic, profile and click sessions) for internet companies (Mohallick et al. 2018).

Unlike CF and CBRS, a Session-Based Recommendation System (SBRS) only takes users' current sessions (i.e., what users clicked and checked in the most recent fixed time) into consideration in making recommendations. It has gained much attention in some conditions (e.g., small online stores) which only involve anonymous data (Zheng et al. 2020). Compared with CF and CBRS, SBRS is cheaper in terms of information retrieval and focuses more on users' current needs. The principle of SBRS is simple, most of the SBRSs take users' current click sessions as input and find patterns from user behavior and finally, output a list of recommended products. Markov chain is one of the state-of-the-art models in SBRS (Shani, Brafman, and Heckerman 2012). It predicts users' next item based on users' previous item views in sessions. In recent years, deep learning models significantly improved the performance of SBRS. Recurrent Neural Networks (RNN)-based models such as GRU4REC (Hidasi et al. 2015) and NARM (Li et al. 2017) can provide more accurate recommendations compared with Markov Chain-based SBRS because of their advantages in processing sequential data. RNN based models generate a representation of a user from the user's click sessions and make recommendations based on that representation. However, RNN-based models still require adequate sessions as input to generate appropriate representation (Wu et al. 2019). If the click session is too short, as these happen a lot in real data, the model will fail to give good recommendations. Another drawback of the naïve RNN-based models is that they neglect long-distance item transitions. It is proved that item transitions can be used as a local factor in SBRS (Li et al. 2017). However, most of the current RNN-based models (e.g., GRU4REC, STAMP) focus more on the relationship of consecutive item transitions (e.g., the relationship between the first item and the second item in a session) and often overlook the relationship of distant item transitions (e.g., the relationship between the first item and the thirteenth item in a session) (Wu et al. 2019).

Graph neural networks (GNN)-based RS models tackle these problems by taking session graphs made by user's anonymous click sessions as input. The sessions used by current RNN-based SBRS are unidirectional and consecutive, while the session graphs are multidirectional and complex in structure (session graphs are composed of nodes and edges, just like chemical formulas are composed of chemical element symbols and dashes). Rather than learning patterns directly from user behaviors and generate user representations like RNN-based models, GNN-based SBRSs generate representations of items from user's item transitions, that is to say, GNN does not need to know the user very well, it gives recommendations based on patterns of item transitions. Therefore, short sessions are still valuable for GNN models and the relationships between distant items are considered. Current researches on GNN-based SBRS focus on generating better item representations and reading out (convert item representations into a fixed-size vector or final output) representations more efficiently. For example, Wu et al. (2019) introduced a Session-based Recommendation System with Graph Neural Networks (SR-GNN) which utilizes an attention network to generate item representations from

neighboring items iteratively. Qiu et al. (2019) introduced a Full Graph Neural Networks (FGNN) with a new readout function which is specifically designed for SBRS. On the current researches of GNN, there are nearly none of them tries to discuss the role of dwell time (i.e., the time that users spend for each item view) in GNN-based SBRS.

The dwell time has been proved to be a valuable feature in RNN-based SBRS since the dwell time is highly related to users' interests and preferences (Dallmann et al. 2017; Bogina and Kuflik 2017; Wang, Li, and Yan 2019). The hypothesis is that users take more time on things they like. The results support this hypothesis, after adding dwell time as a feature in inputs of RNN-based SBRS, the accuracy improved significantly. However, how about adding dwell time as a feature in session graphs and processing the session graphs incorporating dwell times in GNN-based SBRS? The dwell time may reflect some information about the relationship between items and further improve the accuracy of GNN-based SBRS. This hypothesis brings the research question of this thesis: what is the role of dwell time in the performance of GNN based SBRS models? The question can be further divided into three sub-questions:

1. How to incorporate dwell time in session graphs?
2. Whether the dwell time provides any information to explain the relationship of two items in a session?
3. Is the performance of a GNN model incorporating dwell times better than that of a GNN model without dwell time?

The rest of this thesis is organized as follows. Section 2 reviews the related works. Section 3 explains the proposed method. Section 4 introduces the experimental setup. Section 5 illustrates the results of this research. Section 6 discussed the results, and lastly, section 7 draws a conclusion.

2. Related Work

In this section, some related works are reviewed and discussed. We introduce the widely used recommendation systems (RS) in section 2.1, Session-Based Recommendation systems (SBRS) in section 2.2, GNN-based SBRS in section 2.3 and dwell-time in RS in section 2.4. Lastly, we explain the scientific relevance of this research in section 2.5.

2.1 Widely used Recommendation Systems

As mentioned above, most of the recommendation systems being used are either Collaborative Filtering (CF) or Content-Based Recommendation System (CBRS) or the combination of these two (Isinkaye, Folajimi, and Ojokoh 2015). CF is based on the theory that similar people would have similar interests or preferences. It gives recommendations to users by collecting the interests of other related users. For example, if many of your friends watched one specific movie, the system would be very likely to recommend that movie for you as well. CF is mostly adopted when the long-term user history is limited (Lu et al. 2015). CBRS gives recommendations by comparing the similarity of candidate contents with users' previously interested contents. The similarity of contents can be measured by characteristics of contents such as movie genres or book authors (Isinkaye, Folajimi, and Ojokoh 2015).

Compared with CF, CBRS does not require an active social relationship from the user or a substantial number of users from the platform. Although these two methods

achieved satisfactory results in practice, given people's ever-increasing awareness of information security, it is getting harder to acquire users' information (Mohallick et al. 2018). Therefore, the utilization of these two RS is highly restricted.

2.2 Session-based Recommendation Systems

As an important branch of RS, SBRS takes user's recent behavior sequences as input rather than processing all the users' historical behaviors. One of the classic examples of SBRS is Markov Chain. Shani, Brafman, and Heckerman (2012) treat the recommendation process as a sequential optimization problem and utilize the Markov Decision Process (MDP) to decide which item should be recommended next. They also stated that the decision of MDP should aim for optimizing specific criteria (e.g., profit, cost) Inspired by the idea of MDP which treated the recommendation problem as a sequential decision problem, Rendle, Freudenthaler, and Schmidt-Thieme (2010) further introduced Factorizing Personalized Markov Chains (FPMC) model which factorizes personalized probability transition matrix of users. Compared with MDP, FPMC takes the user's personal transition graphs into model and thus yields better accuracy. However, the performances of MDP and FPMC are highly limited because the recommendation generated from Markov Chain strongly depends on the previous item and both of the models ignore the complex latent relationships among items.

Recently, many researchers shifted their attentions from Markov Chain-based models to the RNN models. Hidasi et al. (2015) are the pioneers of RNN-based recommendation system. They introduced the famous GRU4REC which utilizes multiple Gated Recurrent Unit (GRU) layers to process the current sessions in a parallel manner. GRU4REC became the basis of deep-learning application in the recommendation system domain and further clarified the features of SBRS: anonymous, current sessions based and current needs focused. Inspired by GRU4REC and the popular attention layer, NARM (Li et al. 2017) captures users' current needs by using stacked hybrid GRU encoders with attention mechanisms. The self-attention layer assigns a weight for each hidden state of items and adds up all the hidden states to formulate a session embedding. To further improve the performance of RNN-based SBRS in long-term sessions, STAMP (Liu et al. 2018) captures both users' current interests and general interests by the different weights generated from the self-attention layer. Although RNN-based SBRSs achieved good results, their performances are still very sensitive to user sessions' length and quality.

2.3 GNN-based SBRS

Applying GNN models in SBRS is a new trend in the recommendation system domain. Wu et al. (2019) introduced SR-GNN which makes current sessions into session graphs and then uses a self-attention network to represent each session graph into a composition of both global and local preferences of users. They argued that previous researches in SBRS always assumes that there is a representation of user for each session and this prejudice assumption ignores the impacts of item transitions. Wu et al. (2019) used the hybrid embedding of both local interests and long-term interests of the user to represent sessions. Xu et al. (2019) further developed the SR-GNN by adding two additional self-attention networks after the attention networks of SR-GNN. The first added network is the point-wise feed-forward network which endows the model with nonlinearity and measures the inner-connections between latent dimensions. The second added network is a residual connection that leverages the information of low-level latent dimensions.

These two added networks further improved the quality of session representations. Qiu et al. (2019) treated the recommendation problem as a graph classification problem. They introduced a Weighted Graph Attentional Layer (WGAT) in their FGNN model to incorporate the edge weight in the aggregation of neighboring nodes. They also created a new readout function from modifying the Set2Set extractor to give out a representation of global session graphs. STAR-GCN (Zhang et al. 2019) stacks several Graph Convolutional Networks (GCN) layers as an encoder-decoder system and applies a latent user and item embedding technique to maintain the space complexity of model. Although the above researches achieved better accuracy compared with RNN-based SBRS, they neglected the possibility of adding more features in inputs.

2.4 Dwell time in RS

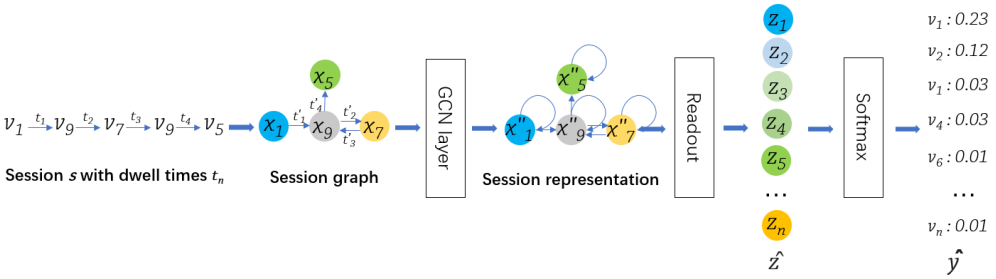
Many researchers attempted to improve the performance of other RSs by adding dwell time. Bogina and Kuflik (2017) set a predefined threshold t and increased the presence of items in that session which have dwell time greater than the threshold (e.g., the original session [1, 2, 3, 4] became [1, 1, 2, 3, 3, 4] because "1" and "3" have dwell times of $2t$). This item Boosting method is based on the theory that the users' interests are positively related to dwell time. Dallmann et al. (2017) computed the dwell time from timestamps and used two separate GRU networks to generate representations of both dwell time embeddings and item embeddings. At each step in the sequence, the represented dwell time and the represented item are concatenated and the result is used as an input of IT-RNN. Dallmann et al. (2017) also emphasized that the dwell time can be very sparse and after adding the dwell time, the computations could become inefficient. These researches proved that adding dwell time as an input feature in RNN-based SBRS can improve the quality of session representation.

2.5 Scientific Relevance

Based on the mentioned researches in other recommendation systems, the RS can be improved by either applying new algorithms or adding new features (e.g., user history, profile, ratings). Current researches on GNN models focused on applying existing algorithms such as GRU and self-attention to generate more accurate item and session representations or making improvements in propagation methods (e.g., FGNN). However, they neglected that dwell time can be used as a new feature in GNN-based SBRS. The dwell time may have different meanings in RNN-based SBRS and GNN-based SBRS. In RNN models, the dwell time reflects users' interest in a product. While in GNN models, the dwell time may relate to the latent pattern of item transition. Before utilizing the dwell time in GNN models, many questions about the roles of dwell time in GNN models are yet to be answered. First, how to incorporate dwell time in session graphs? Second, whether the dwell time provides information to explain the relationship of two items in a session? Third, is the performance of a GNN model incorporating dwell times better than that of a GNN model without dwell time? For answering these questions, the performances of a Graph Convolutional Neural Networks (GCN)-based recommendation system under three different conditions: "occurrence only" (baseline), "dwell time only", "occurrence with dwell time" were compared. The Tooso Fashion Clickstream Dataset (TFCD) which was obtained by recording the user sessions of a European e-commerce fashion website was used. This research sheds light on further researches on time-aware GNN-based recommendation systems.

3. Methods

Figure 1
The pipeline of GCN-based SBRS. The inputs of this model are session graphs made from session sequence *S* and dwell times t_n. After processing the session graph individually, The GCN layer generates a session representation for each session graph. The session representation processed by GCN layer has the same structure as the original session graph except the nodes x_n in the session representation become nodes representations x''_n. By inserting the session representation in the readout function, a z-score is calculated for each $v_n \in V$. Finally, the model gives every item v_n a score $\hat{y}$ which denotes the probability that the item will be the next-click.



In this section, the structure and principle of GCN-based SBRS is explained in detail. Firstly, the problem setting is illustrated in section 3.1. Secondly, the different approaches to scale dwell times is introduced in section 3.2. Thirdly, the approach of making session graphs from users' current sessions is explained in section 3.3. Fourthly, the way of learning session representation from session graphs is described in section 3.4 and lastly, the readout function that generates the probability score $\hat{y}$ is explained in section 3.5. The general pipeline for the whole model is demonstrated in Figure 1.

3.1 Problem Setting

The goal of SBRS is to predict the next item that users intend to click given users' current sessions only (Hidasi et al. 2015). In this research, let the $V = [v_1, v_2, v_3, ..., v_m]$, $m \in \mathrm{R}$ denotes all the products of a commercial website. A current session of an anonymous customer can be represented by $S = [v_{s,1}, v_{s,2}, v_{s,3}, ..., v_{s,n}]$ in which all the items are sorted in time order and $v_{s,n} \in V$ represents an item clicked by the user in sessions *S*. The list of dwell time $T = [t_1, t_2, t_3, ..., v_n]$ denotes the dwell times distribution of *S* and $t_n \in T$ denotes the dwell time spent from item $v_{s,n}$ to item $v_{s,n+1}$. GCN-based SBRS takes the *S* and *T* as input and gives every item in *V* a probability $\hat{y} \in \hat{Y}$ as score. The items with top-*k* scores (i.e., first *k* items in score list $\hat{Y}$ which is sorted from highest to lowest) will be recommended to the user. In short, the task of this research is to fit a model M defined by $\hat{Y} = \mathrm{M}(X)$ to compute the probabilities of all the candidate products of being the next click $\hat{Y}$ from session $X = [v_{x,1}, v_{x,2}, v_{x,3}, ..., v_{x,n}]$. This task can be treated as a multi-class graph classification task from the perspective of types of machine learning tasks.

3.2 Scaling dwell times

Dwell times were calculated from timestamps of user actions (i.e., item views) and therefore measured in milliseconds (ms). The result of Exploratory Data Analysis (EDA) showed that the dwell times in TFCD sessions are very sparse in distribution. As stated by Dallmann et al. (2017) in their research, the spares dwell times with a high resolution are unlikely to be interesting in measuring user interests. Therefore, the dwell times have to be scaled before incorporating them in session graphs.

This research used two scaling methods, normalization and inverse-normalization. These two methods are based on two hypotheses: first, there is a positive correlation between dwell times and relationship strengths (i.e., the shorter the dwell time between two items, the weaker the relationship between them). Second, there is a negative correlation between dwell times and relationship strengths (i.e., the shorter the dwell time between two items, the stronger the relationship between them). The formula for normalization is:

$$t_{norm} = \frac{t - t_{min}}{t_{max} - t_{min}}, t \in T \tag{1}$$

The dwell times normalized by above formula were used to test whether there is a positive correlation. The formula for inverse-normalization is:

$$t_{inverse-norm} = \frac{t_{max} - t}{t_{max} - t_{min}}, t \in T \tag{2}$$

The dwell times normalized by inverse-normalization were used to test whether there is a negative correlation. In these two formulas, t_{max} and t_{min} are the biggest and smallest dwell time in the list. Note the scaling was conducted session-wise, so the t_{max} and t_{min} changed in different sessions.

The dwell times of each session were scaled in both ways and the performances of these two scaled dwell time groups were compared to prove which hypothesis is correct.

Besides the above two methods, the robust scaling which scales the data according to quantile range or median was also considered. Because the outliers in some sessions (i.e., dwell times with unreasonable lengths) can result in an extreme t_{max} or t_{min}, scaling by quantile range or median may provide better results. However, the GCN layer cannot recognize negative numbers as weights, thus, only the normalization scaling and inverse-normalization scaling were tested in this experiment.

3.3 Constructing session graphs

Before making session graphs, it is worth to understand the different types of session graphs.

In the perspective of transition direction, a session graph can be either directed or undirected. In directed session graphs, a transition from $v_{s,n}$ to $v_{s,n+1}$ is different from a transition from $v_{s,n+1}$ to $v_{s,n}$. While an undirected session graph does not record the direction of transitions, a transition from $v_{s,n}$ to $v_{s,n+1}$ is the same as a transition from $v_{s,n+1}$ to $v_{s,n}$.

From the perspective of weighting, a session graph can be either weighted or unweighted. A weighted graph contains an edge weight which indicates the relationship

strength of the two ends of that edge. While an unweighted graph does not record the relationship strength.

In this research, the natural order of item transitions determines that the session graphs should be directed. The dwell time was incorporated on the edge weight, so the session graphs have to be weighted. Consider the above requirements, the constructed session graphs for this research should have three parts, nodes that represent the items, edge indexes that represent the transitions and edge weights that record the weights of edges.

More specifically, let $G_s(V_s, E_s)$ denotes a session graph which made from a session of length n where $v_{s,n} \in V$ represent items' nodes and $(e_{s,n}, e_{s,n+1}) \in E_s$ are edge indexes which mean a user clicks item $v_{s,n+1}$ after $v_{s,n}$. The node $v_{s,n}$ represents the appearance of item n in session S. Remember the appearance of item here is different from occurrence of item. For example, a session list $[v_{s,1}, v_{s,2}, v_{s,1}, v_{s,3}, v_{s,2}]$ will create node list $[v_{s,1}, v_{s,2}, v_{s,3}]$ after graph construction and there should not be any repeated item in the node list of a graph.

After extracting the nodes V_s and edge E_s from the last step, the next step is to compute the edge weights. As mentioned above, this research compared the performances of GCN-based SBRS under three conditions: "occurrence only", "dwell time only" and "occurrence with dwell time". The methods of graph construction under these three conditions are different in edge weight $(w_{s,n}, w_{s,n+1})$ which represents a feature of the edge $(e_{s,n}, e_{s,n+1})$:

- In "occurrence only" group, the $(w_{s,n}, w_{s,n+1})$ denotes the occurrence of a transition from $v_{s,n}$ to $v_{s,n+1}$. For example, $(w_{s,1}, w_{s,2})$ in session $S = [v_{s,1}, v_{s,2}, v_{s,1}, v_{s,3}, v_{s,1}, v_{s,2}]$ should be 2 because the transition from $v_{s,1}$ to $v_{s,2}$ happened twice.
- In "dwell time only" group, the $(w_{s,n}, w_{s,n+1})$ denotes the scaled dwell time $t'_{s,n}$ between $v_{s,n}$ and $v_{s,n+1}$. Similar to the occurrence of transitions in "without dwell time" group, the dwell time of a repeated transition should be recorded cumulatively. For example, if the dwell times of transitions between $v_{s,n}$ and $v_{s,n+1}$ are t'_1 for the first time and t'_2 for the second time, the $(w_{s,n}, w_{s,n+1})$ should be $(t'_1 + t'_2)$.
- In "occurrence with dwell time" group, $(w_{s,n}, w_{s,n+1})$ denotes the general relationship strength calculated from multiplying occurrence and scaled dwell time. For example, if the transition from $v_{s,n}$ to $v_{s,n+1}$ happened twice and the scaled cumulative dwell time is $(t'_1 + t'_2)$, the edge weight $(w_{s,n}, w_{s,n+1})$ should be $2*(t'_1 + t'_2)$. The product of scaled dwell time and occurrence means the occurrence is amplified by an index t' which contains the information about the relationship strength of two nodes.

There are other options to compute the edge weights in "occurrence with dwell time" group. The edge weight of a specific transition in this group could be seen as a compression of both occurrence and dwell time of that transition. The purpose here is to calculate an edge weight $(w_{s,n}, w_{s,n+1})$ which indicates the relationship strength of $(v_{s,n}, v_{s,n+1})$) without loss of information from both occurrence and dwell time. Therefore, this problem could be seen as a dimensionality reduction problem. Several dimensionality reduction methods such as Primary Component Analysis (PCA) and Singular Vector Decomposition (SVD) were tried. However, the lengths of most of the sessions are insufficient to calculate reliable variances, these methods did not provide

any useful edge weight. The detail explanation of these failures is discussed in Section 6.

3.4 Learning session representation from session graphs

As shown in Figure 1., after obtaining the session graphs, the next step is to learn a session representation from each session graph. There are several options for learning session representations. As mentioned in Section 2, SR-GNN and FGNN were applied under a very similar setup. However, we regretfully found that the codes of these two models on GitHub are incomplete and cannot be fixed given the time limit. As suggested by Zhang et al. (2019), GCN is also a suitable layer for SBRS task. Compared with SR-GNN and FGNN, the structure of GCN is simpler and computationally inexpensive. Nevertheless, the simpler structure also means the representation quality may not as good as its counterparts. SR-GNN uses gated graph neural networks to control what information from the neighboring nodes should be reserved in the new node representation (Wu et al. 2019). FGNN uses self-attention mechanisms to determine the importance of each neighboring node for a specific node in node representation generation (Qiu et al. 2019). While GCN generates representation by collecting information from neighboring nodes with the same weight. Because the purpose of this research is not achieving a high accuracy but to explore the impact of dwell time on GNN-based model, the basic Graph Convolutional Networks was used to generate session representation.

Before learning the session representation, each item $v_{s,n}$ was embedded into a latent vector $x'_{s,n} \in R^d$ where d is the dimension of the embedded node by:

$$x'_{s,n} = Embed(v_{s,n}) \tag{3}$$

The reason for embedding the nodes is to convert the meaningless item serial numbers into learnable features.

Now the session graph (node embeddings $x'_s \in R^{n*d}$ where n is the number of nodes, edge indexes E_s and edge weights W_s) for a GCN layer is available. The next step is generating new node representations from their neighboring nodes. The core propagation formula for GCN layer is:

$$x''_s = \hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}x'_sW \tag{4}$$

Where $x''_s \in R^{n*d}$ is the new set of node representations of the original node embeddings $x'_s \in R^{n*d}$. A is the adjacency matrix which indicates the relationship between all nodes in a session graph, in this experiment, the adjacency matrix is the edge weights matrix obtained from last step. $\hat{A}$ is defined as the adjacency matrix A inserted with self-loops I. The insertion of self-loops makes sure that the node itself is also considered when generating the new node representation. $\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$ is the diagonal degree matrix made from edge index which is used to normalize the adjacency matrix. Without normalized by $\hat{D}$, the product of $\hat{A}$ and x'_s will change the scale of x''_s. W$\in R^{n*d}$ is a nodes-specific trainable weight matrix.

For understanding the implication of this formula, this formula can be split into two parts, the symmetric normalized Laplacian $\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}$ and the weighted embedding of nodes x'_sW. The product of these two parts is the feature or the new node representation of each weighted embedding x'_sW, which obtained by applying shared weights

$\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}$ on weighted embedding x'_sW. The function of symmetric normalized Laplacian here is very similar to the kernel in Convolutional Neural Networks (CNN) and that is where the name "Graph Convolutional Networks" comes from. If a Sobel kernel is applied on an image , the edge feature could be extracted from that image. In GCN, the information of neighboring nodes can be extracted by symmetric normalized Laplacian.

However, the symmetric normalized Laplacian is only suitable for undirected graphs (Schlichtkrull et al. 2018). For directed graphs, the information of edge direction will be ignored by symmetric normalized Laplacian. As recommended by Schlichtkrull et al. (2018), the Nonsymmetric Normalized Laplacian is suitable for weighted and directed graphs. The new propagation rule for direct graphs is replacing the symmetric normalized Laplacian with nonsymmetric normalized laplacian:

$$x''_s = \hat{D}^{-1}\hat{A}x'_sW \tag{5}$$

It is worth to know that some researchers also used undirected graphs as inputs in the social network recommendation systems (Yang and Toni 2018; Tofik and Deorankar 2016). Due to the time limit, the performance of GNN-based SBRS which takes undirected graphs with dwell time incorporated as inputs will be explored in future research.

3.5 Readout Function

The readout function used in this research is inspired by the readout function of SR-GNN (Wu et al. 2019). It takes the session representations generated from the last step as input and gives a list of z-scores as output. The list of z-scores is further processed by a softmax layer which provides a probability score $\hat{y}$ for each candidate product.

As mentioned earlier, the readout function of SR-GNN uses users' current interests S_l and global interests S_g to create a hybrid embedding S_h and the output z-scores are the products of the hybrid embeddings and each node representation $x''_{s,n}$ in session S. The local embedding S_l of session S is defined as the last node representation of session S (i.e., For session S = [$x''_{s,1}$, $x''_{s,2}$, $x''_{s,3}$, ..., $x''_{s,n}$], the local embedding S_l should be $x''_{s,n}$). The global interest embedding S_g is computed by aggregating all the node representations with a soft-attention mechanism. Since the information contained by node representations may have different levels of priority, the soft-attention mechanism can be used to improve the quality of global interest embedding. The function of global embedding S_g is:

$$\alpha_i = q^T\sigma(W_1x''_{s,n} + W_2x''_{s,i} + c) \tag{6}$$

$$S_g = \sum_{i=1}^{n} \alpha_i x''_{s,i} \tag{7}$$

Where the α denotes the attention score of node representation $x''_{s,n}$ for session S. $q \in R^d$ and $W_1, W_2 \in R^d$ control the weights of item representation vectors. c is the bias. By aggregate all the products of node representations and their attention scores α, the global interests S_g can be computed.

Now, the local interest S_l and global interest S_g are available. The hybrid embedding S_h can be obtained by applying linear transformation on the concatenation of S_l and S_g:

$$S_h = W_3[S_l; S_g] \tag{8}$$

Where the matrix $W_3 \in R^{d*2d}$ compresses the two interest embeddings together. The z-scores for all the items $v \in V$ can be defined as:

$$\hat{z_i} = {s_h}^T v_i \tag{9}$$

In this function, each candidate item is transposed by the hybrid session embedding S_h. Finally, a softmax function is applied on the list of z-scores $\hat{Z}$ and the output is a list of probabilities of items to be the next-click in session S:

$$\hat{Y} = softmax(\hat{Z}) \tag{10}$$

As mentioned above, the whole task is a multi-class classification task, therefore, the cross entropy was used as the loss function in training. It can be written by:

$$\mathcal{L}(\hat{y}) = -\sum_{i=1}^{m} y_i log(\hat{y_i}) + (1 - y_i) log(1 - \hat{y_i}) \tag{11}$$

Where y is the target item that clicked by user after session S. At last, the Back-Propagation Through Time (BPTT) algorithm was used to train the whole model.

4. Experimental Setup

This section explains the setup of this experiment in detail. The dataset used in this research is introduced in section 4.1. Following this, the programming languages and packages are introduced in section 4.2. Next, the data preprocessing and findings from EDA are discussed in section 4.3. After that, the scaling of dwell times and the construction of session graphs are illustrated in section 4.4. Then, the GCN model and evaluation matrices are explained in section 4.5.

4.1 TFCD dataset

The TFCD dataset used in this research is kindly provided by Tooso, a North American company providing A.I services to the e-commerce industry. TFCD is obtained by recording the user sessions of a European e-commerce fashion website for 20 days (Bigon et al. 2019). After collecting all the 5,433,612 events (i.e., the actions of users), the data were then anonymized by removing the user IDs and other personal information (Requena et al. 2020). The features of TFCD include session IDs, hashed client user IDs, event types which records the types of actions in that event, hashed item IDs, timeframe measured in milliseconds which records the timestamps of user actions and event date in format year-month-day which records the date of user actions.

There are six different types of actions in the events, “view” which triggered when the user visits any page on the website, “detail” which triggered when user visits any description page of a specific product, “add” which triggered when user adds a product in the shopping cart, “remove” which triggered when user removes any

product from the shopping cart, "buy" which simply means the user paid for a product and "click" which triggered when user actively search any specific product by using the e-commerce search bar. Among the six actions, only the action type "view" does not involve interaction with items.

4.2 Programming languages and packages

This experiment was fully implemented on Python 3.7.3. The packages, imported functions and their usages are introduced in Appendix 1.1.

Note the GCN layer from package torch_geometric.nn was modified, the method of modification is explained in section 3.4 and the detail of modification is explained in section 4.5.

4.3 Data Preprocessing

For preparing the data used in making session graphs, first of all, we need to extract sessions from events (the code of Dr. Jacopo Tagliabue inspires this step), that is to say, to convert a series of user actions into a session.

In the beginning, the events with the same session IDs were grouped and sorted in time order. Then, four empty lists were created to store the actions, dwell times, item IDs and timestamps of events in a specific session. A "for" loop is used to get the above information from events and insert them into corresponding lists. After this extraction step, the 5,433,612 events formulated 203,958 sessions. An example session can be found in Appendix 1.2.

Secondly, extracting the information needed from sessions. For this research, only the sessions contain an action type of "detail" was used since the checking of products reflect users' interests. Although the "click" also reflects that the user is interested in some products, the weights of interests are different in "detail" and "click", that is to say, users have stronger interests on the products they actively searched than that of products they checked from a list of products. If action "click" is also added in session graphs, the session graphs will be imbalance in weights.

From the last step, a session list was acquired. Because only the item transition orders and dwell times matter in this research, two new lists, "sequences" and "times" were created to store the "item ids" and "dwell times" of each session. Again, a "for" loop was used to extract the events with an "action" of "2" (remember the "2" means "detail") from each session. The events' "item ids" and "dwell times" were recorded in "sequences" and "times" respectively.

Thirdly, cleaning the remaining data. In this step, following procedures were conducted:

1. Deleted empty lists in "sequences" and "times".
2. Deleted the items which appeared less than 5 times in all sessions and their corresponding dwell times.
3. Deleted sessions which only have one item.

The detailed procedures of data cleaning can be found in Appendix 1.3. After data cleaning, the lengths of "sequences" and "times" were both 160,291.

Fourthly, augment the remaining data. As suggested by other papers (Tan, Xu, and Liu 2016; Wu et al. 2019; Qiu et al. 2019), data augmentation can

be used in the training of RSs to increase the amount of data. In this research, "sequences" and "times' were augmented by splitting lists of length n into $n-1$ lists. To be specific, $S = [v_1, v_2, v_3, v_4, \ldots, v_n]$ will be partitioned into $([v_1], v_2), ([v_1, v_2], v_3), ([v_1, v_2, v_3], v_4), \ldots, ([v_1, v_2, v_3, v_4, \ldots, v_{n-1}], v_n)$ where the last item is the target y of each sequence (i.e. v_1, v_2, v_3, v_4 and v_n in this example). To record the targets of each augmented sessions, a new list named "targets" was created. The "times" was augmented in the same way as "sequences" except the target y was not recorded. Now, there are three lists, "sequences", "times" and "targets" available, which represent the augmented sessions, the augmented dwell times and the targets y respectively. Every list has 1,277,321 lists inside.

This augmentation brought some sessions with lengths less than 2 and these sessions cannot be used to make session graphs, hence, these sessions were deleted by applying "for" loop. Afterwards, we also found that some lists in "sequences" only have one item involved but repeated several times (e.g., $[[504B1A7347A89D42], [504B1A7347A89D42], [504B1A7347A89D42]]$). These sessions only have self-transition but no transitions between items, therefore, these lists and corresponding "times" and "targets" were deleted by checking the number of unique items for all the lists in "sequences". The function "unique ()" which created for finding the number of unique items can be found in appendix 1.4. Those lists in "sequences" with a unique element of less than 2 and the corresponding lists in "times" and "targets" were filtered out. Finally, after shuffling these lists in triples by random.shuffle, there were 959,830 lists left for "sequences", "times" and "targets". The reason for shuffling here is to make the data more balanced in length.

4.4 Scaling dwell times and constructing session graphs

As mentioned in section 3.2, the dwell times were scaled in two methods which were defined as two functions in Python, "normalization()" and "inverse_normalization()" (Appendix 1.5). The input of normalization is the list "times" which contains 959,830 lists. The length of output does not change after scaling.

The next step is to split the data. The "sequences", "times" (with two methods scaling) and "targets" were split parallelly into training, validation and testing set with proportions of 90%, 5% and 5%. The training set in this task is bigger than usual machine learning tasks because this is a multi-class classification task with thousands of different labels, as suggested by many researchers (Tan, Xu, and Liu 2016; Wu et al. 2019; Qiu et al. 2019), the average proportion of the training set for RSs is around 90%.

Now, all the necessary data for making session graphs are ready: the session sequences "sequences", the normalized dwell times "times" and the label "targets". They were used to make three types of session graphs: "occurrence only", "dwell time only" and "occurrence with dwell time". After scaling the dwell times with two methods, there were in total five types of session graphs, the "occurrence only" (O), "normalized dwell time only" (N-T), "inverse-normalized dwell time only" (I-N-T), "occurrence with normalized dwell time" (O & N-T) and "occurrence with inverse-normalized dwell time" (O & I-N-T).

A function "making_graphs" was created for making these types of session graphs. It takes the "sequences" list, "targets" list, "times" list, a string which represents the name of graph types and a boolean named "shuffle" to indicate whether to shuffle the graphs when making batches as inputs. This function first assert the name of the graph types is one of the "occurrence only", "dwell time only" and "occurrence and dwell time". Then for each type of graphs, a unique set of codes was used. These three sets

of codes are very similar in structure (Appendix 1.6). The outputs of "making_graphs" were five groups of graphs, O, N-T, I-N-T, O & N-T, and O & I-N-T which all have a batch size of 32. Note the batch size should always be set to the power of 2 because we are going to use GPU to make the training faster and GPU works the most efficiently when the batch size is the power of 2. The number 32 was determined by observing the performances of different batch size on the validation set.

4.5 GCN model and evaluation matrices

The whole GCN model contains three parts, the main layers "GCN_model", the z-score calculator "embedding2score" and the training machine "forward".

"GCN_model": The "GCN_model" takes the hidden size and number of nodes as imputs, It has two layers, the first layer is embedding layer which creates node embedding $x'_{s,n}$ for each node. the second layer is a GCN layer which generates session representations $x''_{s,n}$ from $x'_{s,n}$. The detail of "GCN_model" can be found in appendix 1.7. The outputs of this GCN layer are the session representations, as described in section 3.4.

"embedding2score": The session representations and nodes embeddings $x'_{s,n}$ generated from "GCN_model" will be the inputs of "embedding2scores" which gives a list of z-scores as output. Similar to "GCN_model", "embedding2scores" inherits the torch.nn.module and has a constructer "__init__" to define the hidden size of weights (q, W_1, W_2, W_3) in equation (6) and equation (8).

A function "forward" uses the session representations and nodes embeddings $x'_{s,n}$ to calculate the z-score by equations (9) which is explained in section 3.5.

"forward": The function "forward" takes the set model, the dataset, the top_k, the optimizer and a Boolean named "train_flag" which indicates the mode of model (train, validate or test) as inputs. In the "train" mode, the model will be trained by the dataset and gives the average learning loss per epoch as output. In the "validate" or "test" mode, the model will be evaluated by the dataset and gives the Precision@20 and MRR@20 (Mean Reciprocal Rank of top 20 items) as outputs (Appendix 1.8).

Same as other GNN models (Tan, Xu, and Liu 2016; Wu et al. 2019; Qiu et al. 2019), the evaluation matrices used in this research are Precision@20 and MRR@20. Precision@20 calculates the proportion of top 20 items in which the recommended item y is relevant in the top 20 rankings, and MRR@20 is the normalized ranking of relevant items in the top k ranking.

More specifically, if the target y is in the "sub_scores", a "1" will be appended in the "hit" list, a "1/ rank of y" will be appended in "mrr" list. If the target y is not in "sub_scores", a "0" will be appended in both "hit" and "mrr". Lastly, use "np.mean()" on both "hit" and "mrr" to get the average performances of model. These two values can be computed by the following functions:

$$Precision@K = \frac{n_{hit}}{N} \tag{12}$$

where the n_{hit} represents the number of times that the target y is in the "sub_scores" and N represents the total number of tests.

$$MRR@K = \frac{1}{N} \sum_{v_{label} \in S_{test}} \frac{1}{Rank(v_{label})} \tag{13}$$

where v_{label} represent the target y.

The optimization method used in this task was *Adam* with a learning rate of 0.007 for O graphs and a learning rate of 0.005 for other graphs. The reason for using different learning rates is explained in Section 5.

A learning rate scheduler "StepLR" from Pytorch was used to decay the learning rate by 10% every three epochs. These two parameters were set by observing the performances of different hyperparameters on the validation set.

5. Results

The overall results of this research are shown in Table 1. Among all the five graph types, the baseline "Occurrence only" graphs performed the best in GCN-based SBRS with a Precision@20 of 26.79% and MRR@20 of 8.48%. The proposed dwell time incorporated session graphs did not outperform the baseline. In the rest of this section, three research questions are answered based on the overall results. Lastly, some insights about learning rate selection are explained.

Table 1
The performance of GCN-based recommendation system with different graph types.

Types of session graphs	Precision@20	MRR@20
Occurrence only(O) (baseline)	**26.79**	**8.48**
Normalized dwell time only(N-T)	5.64	1.34
Inverse-normalized dwell time only(I-N-T)	**24.55**	7.80
Occurrence with normalized dwell time(O & N-T)	5.48	1.27
Occurrence with inverse-normalized dwell time(O & I-N-T)	23.83	7.46

5.1 Results of research questions

Q1: How to incorporate dwell time in session graphs?

In this research, the dwell times were incorporated in session graphs by multiplying the occurrence of item transitions and the normalized dwell times. For further exploring the correlation between dwell times and relationship strengths of items, two normalization methods were used: normalization and inverse-normalization. It is obvious from the Table 2. that "inverse-normalized dwell time only" graphs which had a Precision@20 of 24.55% and an MRR@20 of 7.80% performed much better than "normalized dwell time only" graphs which only had 5.64% in Precision@20 and 1.27% in MRR@20. This result showed that there is a negative relationship between dwell times and relationship strengths, in other words, the shorter the dwell time, the stronger the relationship between the two items in an item transition.

Q2: Whether the dwell time provide any information to explain the relationship of two items in a session?

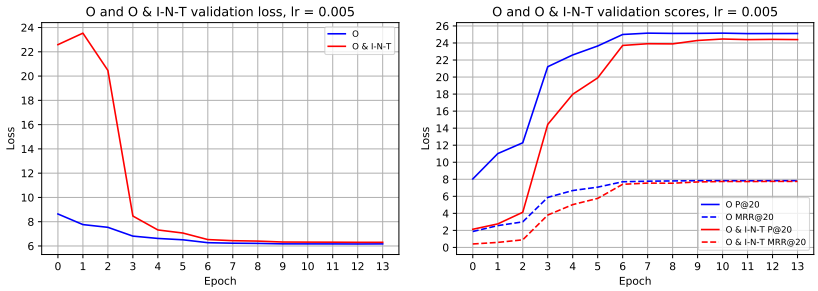
The performance of I-N-T graphs showed that dwell time does provide information to explain the relationship of items. However, compared with the baseline, O graphs (26.79%, 8.48%), both the Precision@20 and MRR@20 (24.55% and 7.80%) are slightly lower.

Q3: Is the performance of a GNN model incorporating dwell times better than that of a GNN model without dwell time?

Unfortunately, the Precision@20 and MRR@20 of O I-N-T graphs were 23.83% and 7.46% respectively which are slightly lower than the baseline graph type O (26.79% in Precision@20 and 8.48% in MRR@20). This result showed that after adding dwell times in session graphs, the quality of recommendation decreased. It also showed that the hybrid graphs O & I-N-T did not reserve the information from both occurrences and dwell times properly.

5.2 Learning rate selection

Figure 2
The loss (left) and scores (right) of O graphs and O & I-N-T graphs when the learning rate (lr) is 0.005.

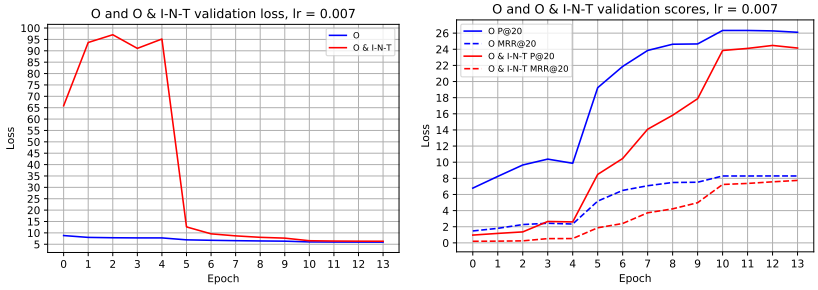


When tuning the hyperparameters, we found that the learning rate could significantly affect the performances of different types of graphs on GCN-based SBRS.

From figure 2 we can see that the validation loss of O graphs stabled at epoch 6 when the learning rate was 0.005 with 10% decay every three epochs. O & I-N-T graphs converged a bit slower at epoch 9.

In terms of performance, the O graphs hit the highest Precision@20 (25.1%) and MRR@20 (7.9%) at epoch 10 and started overfitting after that, while O & I-N-T graphs' Precision@20 and MRR@20 also peaked at epoch 10 with a percentage of 24.2% and 7.85% respectively.

Figure 3
The loss (left) and scores (right) of O graphs and O & I-N-T graphs when the learning rate (lr) is 0.007.



From figure 3. We found that the loss of O & I-N-T graphs was volatile in the first four epochs when the learning rate was 0.007 and dramatically decreased to around 13 at epoch 5. Compared with the validation loss of O & I-N-T graphs in Figure 2, started

training with a learning rate of 0.007 with 10% decay every three epochs was too high for O & I-N-T graphs to learn anything. For O graphs, the loss started from around 8 and gradually decreased to 6.15, which is smoother compared with figure 2.

Compared with the performances in figure 2, we found that both the Precision@20 and MRR@20 of O graphs achieved higher scores (26.2% and 8.15% respectively), which means when the learning rate is too low, the loss of O graphs may get stuck somewhere and fail to reach its optimal performance.

In general, we should use different learning rates to train different types of graphs. More specifically, a higher learning rate should be used on O graphs to learn smoothly and avoid to get stuck during training. While a lower learning rate should be used on O & I-N-T graphs to help the model converge faster.

6. Discussion

The goal of this research is to explore the role of dwell time in the performance of GNN-based SBRS. In general, the experiments show that the dwell times do contain information that can be used to explain the relationship of items in GNN-based recommendation system. The results also show that there is a negative correlation between dwell times and the relationship of items. However, the results failed to prove that the performance of GNN-based recommendation system can be improved by adding dwell times with the proposed method (multiplying occurrence and dwell times). In the rest of this section, some implications and limitations of GCN model and results are discussed in detail.

6.1 GCN model

Compared with other GNN-based SBRS, the optimal performance of the GCN-based SBRS in this research was relatively poor. SR-GNN had a Precision@20 of 70.57% and an MRR@20 of 30.94% on a widely used dataset, Yoochoose (Wu et al. 2019). F-GNN reached a Precision@20 of 71.12% and an MRR@20 of 31.68% on Yoochoose (Qiu et al. 2019). The causes of poor performance could be:

1. The performances of recommendation systems varied significantly on different datasets. For example, the performance of SR-GNN in both Precision@20 and MRR@20 dropped to 50.73% and 17.59% on another dataset, Diginetica (Wu et al. 2019). One of the possible reasons is that the relationship strengths of items in different industries are different. For example, the relationship strengths of items in the hardware industry may stronger than the food industry because most customers visit online hardware stores with a strong intention while the customers of online food stores may not have a clear plan about what foods they want.
   Besides, the varied performances of the same model on different datasets can be explained by the different methods of data collection and data cleaning. Due to the time limit, we did not use other datasets to test the performance of GCN-based SBRS.
2. The quality of session representations generated by GCN layer may not as good as the session representations generated by SR-GNN or F-GNN. As mentioned above, the GCN layer gathers information from neighboring nodes with same weights while SR-GNN and F-GNN use gated graph

neural networks or attention mechanism to generate more accurate session and node representations.

6.2 Research questions

Q1: How to incorporate dwell time in session graphs?

In this research, several methods (multiplication, PCA and SVD) were tried to reduce the edge weights from two-dimensions matrices (occurrences and dwell times of transitions) into a one-dimension matrix. Among them, only the multiplication method generated meaningful results. The other two options failed because the dwell times of some sessions are sparse, short and full of noise (e.g., customers may leave their computers and generate long dwell times which are meaningless for GNN models) while the occurrences of sessions are short with very low variances (most transitions only happened once in a session). The different characteristics of occurrences and dwell times make the results of dimensionality reduction lose most of the information from both matrices and become full of noise.

Although the products of occurrences and dwell times are meaningful, the performances of O & I-N-T graphs and I-N-T graphs show that much information from occurrences and dwell times lost in the multiplication. In this research, the weights of occurrences and dwell times were the same in multiplication. In further research, the multiplication could be done in different weights. For example, multiplying occurrences by 70% of dwell times.

Q2: Whether the dwell time provide any information to explain the relationship of two items in a session?

In earlier researches of dwell time in RNN-based SBRS, the dwell time was proved as positively related to the customer's interest in a product. For example, Bogina and Kuflik (2017) extended the lengths of sessions according to the dwell times of items (i.e., products with long dwell times were copied so that the weights of these products are increased in that session) and successfully improved the performance of RNN-based SBRS. While in GNN-based SBRS, we proved that the dwell time is negatively related to the relationship strength between two items.

From the performance of I-N-T graphs (Figure 1.), the information contained by dwell times was almost as much as the information contained by occurrences. Considering the negative correlation between dwell times and relationship strengths, the information contained by dwell time could be explained as the latent relevance among items. For understanding the information in dwell times, let us assume that most customers visit an e-commerce website with one or more general interests. Customers will keep looking for the next item that may fulfil their current needs until their current needs are fulfilled or they give up. Customers use the dwell time to learn about the product and make decisions. When the two things are fulfilling the same needs or having a strong relationship strength, the time it takes to gather information and make decisions is short. When the two things are fulfilling different needs, customers most likely need more time to learn and think.

For example, John is a reasonable and normal customer who is looking for a T-shirt and a pair of sneakers on an e-commerce website. He decided to start with T-shirts first then looking for sneakers. John does not have a very clear preference on T-shirt, so it takes time for him to read the product descriptions. While he is reading and thinking, John's interest in T-shirt is converging, which means he gradually knows what kind of T-shirts he wants while browsing. So, the time he takes for thinking and making decisions is decreasing. Finally, he bought a T-shirt and start looking for sneakers. Again, John

is not very clear about what sneakers he wants, so he starts to read the descriptions carefully again. Note that from the last T-shirt to the first pair of sneakers, John used a longer time because he is not familiar with sneakers and needs more time to read and think.

Of course, John cannot represent every user, but he could be seen as an averaged user. From the results of experiments, the example of John could be a possible explanation for the negative correlation between dwell times and relationship strengths.

Q3: Is the performance of a GNN model incorporating dwell times better than that of a GNN model without dwell time?

In this research, the result shows that after incorporating dwell times in edge weights by multiplication, the performance of GCN-based SBRS decreased. Possible reasons could be:

1. The edge weights computed from the multiplication of occurrences and dwell times are still full of noises. The PCA and SVD try to maintain as much as information including noises from higher dimensions to lower dimensions, therefore, the noises from dwell times have been reserved in the edge weights. For acquiring good edge weights, one possible solution is applying auto-encoder on occurrences and dwell times. An auto-encoder can reduce dimensionality while denoising the data. For example, Salehi and Davulcu (2019) use graph auto-encoder to learn node representation by reducing the dimensionality of node features. Peng, Guan, and Shang (2019) use auto-encoder to reduce the dimensionality of gene vectors. Gondara (2016) use auto-encoder to denoise medical image. In future research, auto-encoder could be used to reduce the dimensionality of occurrences and dwell times matrices and denoise the result at the same time.
2. GCN layer is not fully capable to generate high-quality session representations. As mentioned above, GCN generates representations by collecting information from neighboring nodes with the same weight. However, without the accessibility to other GNN layers, we failed to test the performance of dwell times incorporated session graphs on other layers.

In the previous GNN-based SBRS (SR-GNN, FGNN, STAR-GNN), the occurrence of item transition was used as a direct and intuitive feature to determine item transition pattern. In this study, we proved that dwell time is also a valuable feature for explaining item transition pattern. Although we did not successfully improve the performance of GNN model by incorporating dwell times, the results show that it is meaningful and possible to be used to generate more appropriate and accurate recommendations. The recommendations with good quality can create value for e-commerce businesses in varies ways:

1. It helps e-commerce businesses maintain a healthy churn rate (i.e., the abandonment rate of e-commerce services) and increase the customer stickiness (i.e., the degree of customer loyalty) (Gómez-Uribe and Hunt 2015).
2. It can increase user engagement and user retention which can be directly translated into business value (Jannach and Jugovac 2019).

3. It can be used to advertise certain products to target customers according to e-commerce companies' needs. For example, persuade customers to choose items with a higher revenue margin (Jannach and Jugovac 2019).

We believe our findings provide valuable basis and experiences for incorporating dwell times in GNN-based SBRS in the future works.

7. Conclusion

The RS can improve users' shopping experience and increase the competitiveness of online stores. Recently, GNN-based SBRS achieved better results than the state-of-the-art RNN-based SBRS. Previous researches on GNN-based SBRS focus on generating better node and session representation. This thesis explores the role of dwell times in GNN-based SBRS by comparing the performances of dwell times incorporated session graphs with session graphs without dwell times. For determining the types of correlation between dwell times and relationship strengths of items, the dwell times are scaled by both normalization and inverse-normalization methods. The results show that there is a negative correlation between dwell times and relationship strengths of items. However, the experiment fails to prove that adding dwell times can improve the performance of GNN-based SBRS. Future research could focus on exploring a better method to compress occurrences and dwell times together or a more effective layer to generate high-quality session representations from dwell time incorporated session graphs.

References

Bigon, Luca, Giovanni Cassani, Ciro Greco, Lucas Lacasa, Mattia Pavoni, Andrea Polonioli, and Jacopo Tagliabue. 2019. Prediction is very hard, especially about conversion. predicting user purchases from clickstream data in fashion e-commerce.

Bogina, Veronika and T. Kuflik. 2017. Incorporating dwell time in session-based recommendations with recurrent neural networks. In *RecTemp@RecSys*.

Dallmann, Alexander, Alexander Grimm, Christian Poelitz, Daniel Zoller, and Andreas Hotho. 2017. Improving session recommendation with recurrent neural networks by exploiting dwell time.

Goldberg, David, David Nichols, Brian M. Oki, and Douglas Terry. 1992. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70.

Gondara, Lovedeep. 2016. Medical image denoising using convolutional denoising autoencoders. pages 241–246.

Gómez-Uribe, Carlos and Neil Hunt. 2015. The netflix recommender system. *ACM Transactions on Management Information Systems*, 6:1–19.

Hemp, Paul. 2009. Death by information overload. *Harvard business review*, 87:82–9, 121.

Hidasi, Balázs, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks.

Isinkaye, Folasade, Yetunde Folajimi, and Bolanle Ojokoh. 2015. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16.

Jannach, Dietmar and Michael Jugovac. 2019. Measuring the business value of recommender systems.

Leimstoll, Uwe and Henrik Stormer. 2007. Collaborative recommender systems for online shops. page 156.

Li, He, Lu Yu, and Wu He. 2019. The impact of gdpr on global technology development. *Journal of Global Information Technology Management*, 22:1–6.

Li, Jing, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural attentive session-based recommendation. pages 1419–1428.

Liu, Qiao, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. 2018. Stamp: Short-term attention/memory priority model for session-based recommendation. pages 1831–1839.

Lu, Zhongqi, Zhicheng Dou, Jianxun Lian, Xing Xie, and Qiang Yang. 2015. Content-based collaborative filtering for news topic recommendation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 217–223, AAAI Press.

Mohallick, Itishree, Katrien De Moor, Özlem Özgöbek, and Jon Gulla. 2018. *Towards New Privacy Regulations in Europe: Users' Privacy Perception in Recommender Systems: 11th International Conference and Satellite Workshops, SpaCCS 2018, Melbourne, NSW, Australia, December 11-13, 2018, Proceedings*.

Peng, Jiajie, Jiaojiao Guan, and Xuequn Shang. 2019. Predicting parkinson's disease genes based on node2vec and autoencoder. *Frontiers in Genetics*, 10.

Qiu, Ruihong, Jingjing Li, Zi Huang, and Hongzhi Yin. 2019. Rethinking the item order in session-based recommendation with graph neural networks. pages 579–588.

Rendle, Steffen, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. pages 811–820.

Requena, Borja, Giovanni Cassani, Jacopo Tagliabue, Ciro Greco, and Lucas Lacasa. 2020. Shopper intent prediction from clickstream e-commerce data with minimal browsing information. *Scientific Reports*, 2020:16983.

Salehi, Amin and Hasan Davulcu. 2019. Graph attention auto-encoders.

Schlichtkrull, Michael, Thomas Kipf, Peter Bloem, Rianne Berg, Ivan Titov, and Max Welling. 2018. *Modeling Relational Data with Graph Convolutional Networks*.

Scholl, Margit. 2018. Scholl, m. (2018 in press). information security awareness in public administrations (provisional chapter) in: Ubaldo comite, public management and administration. open access: Intech d.d.o. rijeka (intechopen) - will be published in june 2018 as open access (https://www.intechopen.com).

Shani, Guy, Ronen Brafman, and David Heckerman. 2012. An mdp-based recommender system. *Journal of Machine Learning Research*, 6.

Tan, Yong Kiam, Xinxing Xu, and Yong Liu. 2016. Improved recurrent neural networks for session-based recommendations.

Tofik, Ranj and Anil Deorankar. 2016. Friend recommendation system based on lifestyles of users. pages 682–685.

Topping, Colin. 2017. *The role of awareness in adoption of government cyber security initiatives - a study of SMEs in the UK*. Ph.D. thesis.
Verhoef, Peter, Wagner Kamakura, Carl Mela, Asim Ansari, Anand Bodapati, Peter Fader, Raghuram Iyengar, Prasad Naik, Scott Neslin, Baohong Sun, Michel Wedel, and Ron Wilcox. 2005. Choice models and customer relationship management. *Marketing Letters*, 16:279–291.
Wang, Mei, Weizhi Li, and Yan Yan. 2019. Time-weighted attentional session-aware recommender system.
Wu, Shu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-based recommendation with graph neural networks. volume 33.
Xu, Chengfeng, Pengpeng Zhao, Yanchi Liu, Victor Sheng, Jiajie Xu, Fuzhen Zhuang, Junhua Fang, and Xiaofang Zhou. 2019. Graph contextualized self-attention network for session-based recommendation. pages 3940–3946.
Yang, Kaige and Laura Toni. 2018. Graph-based recommendation system. pages 798–802.
Zhang, Jiani, Xingjian Shi, Shenglin Zhao, and Irwin King. 2019. Star-gcn: Stacked and reconstructed graph convolutional networks for recommender systems. pages 4264–4270.
Zheng, Yujia, Siyi Liu, Zekun Li, and Shu Wu. 2020. Dgtn: Dual-channel graph transition network for session-based recommendation.

1. Appendices

1.1 Packages used in experiments

Table 2
the packages used in this experiment, the functions used from packages and their usages.

Packages	Functions	Usages
csv	DictReader	reading csv file
os	environ	setting seed
operater	itemgetter	sorting dictionary by values
numpy	average, mean, random, isin, where	working as basic math operator
torch	tensor, manual_seed	making tensors and setting CUDA seed
collections	itemgetter	counting frequencies of list elements
math	sqrt	getting square root
random	seed, shuffle	setting seed and shuffling the lists
torch_geometric.data	Data, DataLoader	making session graphs and session graph batches
torch.nn	Linear	applying linear transformation
torch_geometric.nn	GCNConv	getting GCN layer
datetime	datetime	getting system time

1.2 An example of extracted session from raw TFCD

- Actions list: one of the actions in "view", "detail", "add", "remove", "buy" and "click" labeled by number "1", "2", "3", "4", "5" and "6" respectively.
- Dwell times, calculated by the timestamp of next event minus the timestamp of current event in milliseconds. The last event of a session will have a dwell time of 0.
- Item ID: the involved item ID of current event in a list. For those events without an item involved, this list will be an empty list.
- Timestamp: the epoch timestamp of event in milliseconds

A complete session is a list of four dictionaries which have keys of "action", "dwell time", "item id", "timestamp". The values of keys are the concatenations of lists of all events which created above. In Python, an example session should looks like ["actions": $[1, 2, 1]$, "dwell times": $[t_2 - t_1, t_3 - t_2, 0]$, "item ids": $[[], [504B1A7347A89D42], []]$, "timestamps": $[t_1, t_2, t_2]$].

1.3 The detailed procedures for data cleaning

Step 1: Some empty lists are found in the "sequences' from EDA, therefore, these empty lists in "sequences" and their paired dwell times lists in "times" will be deleted from both "sequences" and "times" by applying "for" loops. After deleting empty elements, check the completeness of both lists again by counting the numbers of empty lists and NAs in both lists, if all the results are 0, it means the dataset is complete. After deleting those empty lists, there are 184,217 lists left in both "sequences" and "times".

Step 2: According to the recommendations of previous researches (Wu et al. 2019; Qiu et al. 2019; Dallmann et al. 2017), the events with item appeared less than 5 times should be removed from sessions as these items may negatively affect the prediction accuracy. For doing so, use collections.counter to count the frequency of each item for all lists in "sequences". Create a "minority" list to store the IDs of items appeared less than five times. Check each list in "sequences" to see whether they contain any item in the "minority" list by "for" loop and delete the items which appeared less than five times from the lists in "sequences" and also their paired dwell times in "times". Then, use the collections.counter again to make sure all the minorities are deleted. Note that this step will not affect the lengths of both lists since the deleted substances are not lists but elements in the lists of "sequences" and "times".

Step 3: Now, all the items appeared are frequently checked items, but some lists in "sequences" are too short to make them into session graphs, thus, we need to get rid of those lists which only have one element by "for" loops.

1.4 The function Unique() for finding numbers of unique items in a session

To find the number of unique items, the "unique()" function is defined to return the number of unique elements in the list (e.g., unique([1,1,2,1,3,5,5] will return 4)).

```
Pseudocode:
Function: unique ("session"):
  newlist => a list
  For every list in "session":
    If the list is not in newlist:
        Append the list in newlist
  Return the length of newlist.
```

1.5 normalization() and inverse_normalization()

"normalization()" finds the maximum and minimum numbers in each list and calculate the normalized dwell time for each element by (x - minnum)/(maxnum-minnum+0.001) where x is each dwell time in a dwell time list, (the 0.001 is used to prevent some lists have the same maximum and minimum numbers, as the normalized dwell times will be rounded off to 2 decimal, it will not affect the results.).

```
    newtime => a list
    for every list in time:
        timelist => a list
        maxnum => the maximum number in the list
        minnum => the minimum number in the list
        for every number in the list:
            content => round the result of (number - minnum)/(maxnum-minnum) to 2
decimals.
            If content >= 0:
                timelist append content
            else:
                timelist append 0
        newtime append timelist
    return newtime
```

The “inverse_normalization()” is very similar to “normalization()” except the formula used to scale dwell times becomes (maxnum-x)/(maxnum-minnum+0.001).

1.6 making_graphs()

```
Function making_graphs (session sequences, session labels, session dwell times, graph type, whether to shuffle):
    Assert if graph type is one of the “no_time”, “with_time”, “time_only”
    If graph type is no_time:
        Datalist => a list
        For session and labels in pair of session sequences and session labels:
            I is 0
            Nodes => a dictionary
            Senders => a list
            X => a list
            For node in session:
                If node not in nodes:
                    Nodes append node as key and I as value
                    X append Nodes
                    I increase by 1
                Senders append value of node in dictionary nodes
            Receivers => a copy of senders

            If len(senders) not equal to 1:
                Delete the last value of senders
                Delete the first value of receivers

            Pair => a dictionary
            Sur_senders => a copy of senders
            Sur_receivers => a copy of receivers
            I = 0
            For sender and receiver in pairs sur_senders and sur_receivers:
                If string of sender and string of receiver in pair
                    Pair increase the value “string of sender and string of receiver” as key by 1

                    Delete the Ith element in senders
                    Delete the Ith element in receivers
                Else:
                    Pair append “string of sender and string of receiver” as key and 1 as value
                    I increase by 1

            Edge attribution => a tensor of every “string of sender and string of receiver” in pairs
            Edge index => a tensor of senders and receivers pairs
            X is the node representation
            Session graph => dataloader (X = X, Edge attribution = Edge attribution, Edge index = Edge index)
```

For “dwell time only” graphs, the “pair” stores the cumulative dwell times instead of cumulative occurrence in “occurrence only” graphs. For “occurrence and dwell time”,

the “pair” stores the products of cumulative occurrences and cumulative dwell times. Other settings and structures are all the same as “occurrence only” graphs.

“GCN_model” is a class which inherits torch.nn.module as the layer holder. Inside “GCN_model”, A constructer “__init__” is used to set parameters for layers. The first layer is the nn.embedding which creates node embedding $x'_{s,n}$ for each node. The parameters of the embedding layer are:

1. the number of nodes (should be 10,850 as there are 10,850 unique items) and
2. the hidden size for each node embedding (after tuning in the validation set, the most appropriate hidden size is 125).

1.7 "GCN_model"

The second layer is a GCN layer which generates session representations $x''_{s,n}$ from $x'_{s,n}$. The two parameters of the GCN layer are input and output shape which should be the same as the hidden size of the embedding layer (i.e., both of them should be 125). Remember that some modifications have to be made before using the original GCN layer provided by Pytorch_geometric on this task, inside the script of “gcn_conv”, change the “deg.pow(-0.5)” into “deg.pow(-1)” and also change the return from “deg_inv_sqrt[row]*edge_weight* deg_inv_sqrt[col]” into “deg_inv_sqrt[row]*edge_weight” (check the explanation for doing this in equation (5)).

A function “forward” is defined to design the inputs of layers in “GCN_model”. The structure of “forward” is the same as the constructer, for the embedding layer, the inputs are the nodes “x” and the outputs are embedded nodes. For the GCN layer, the inputs are

1. embedded nodes generated from the embedding layer.
2. edge index.
3. edge weight.

1.8 two modes of "forward"

In the “train” mode, the model will be trained by the dataset. More specifically, before training, the gradients of all layers and optimizers will be set to zero. Then, each session graph will be fed to the model separately. The scores of trainings and the targets will be used to calculate the loss. At last, the Back-Propagation Through Time (BPTT) will be implemented to update the parameters of the whole model.

In the “validate” or “test” mode, the model will be evaluated by the dataset. Before evaluating, two lists named “hit” and “mrr” are created to record the performance of the model. The top_k items from evaluation results will be stored in a list “sub_scores”.