CSGCN - Context and Side-Information in GCNs

Andreas Stenshøj, Daniel Moesgaard Andersen, Rasmus Bundgaard Eduardsen

Aalborg University

astens16@student.aau.dk dand16@student.aau.dk reduar16@student.aau.dk

June 10, 2021

Abstract

Graph-convolutional neural networks are growing increasingly popular, but most of them limit themselves to simply considering user-item interactions, even though additional information is often available in datasets such as context and side-information. In this paper, we present two ways to incorporate side-information and contextual information into the prediction model of a graph-convolutional neural network named CSGCN-IS and CSGCN-ADJ. Including this additional information allows us to not only improve density of the graph structure, but also to generate recommendations for a specific context that the user is currently in. We empirically evaluate the models in both a context-specific setting as well as a non-context-specific on four different real-world datasets, comparing with several relevant GCN and FM models. The non-context specific evaluation employs an 80-20% training and test data split, and shows improvements in performance from 0.07% − 10.01%*, as well as a decrease on certain datasets of up to* 5.09%*. The context-specific evaluation shows both significant improvements and decreases. An ablation study is also conducted, showing that the inclusion of context and side-information for CSGCN-ADJ does little to improve performance for the non-context specific setting. For the context-specific setting, the ablation study shows that the performance of CSGCN-IS increases when context and side-information are included, whereas CSGCN-ADJ sees little difference.*

1. Introduction

Recommender systems (RS) have become an important part of everyday life on various online platforms.
With the tremendous amount of information available to users, finding what you need without help can be overwhelming. One way RS have attempted to aid users is through collaborative filtering (CF), where the preferences and similarities between users and items are used to generate recommendations. Graph Convolutional Network (GCN)-based models have gained high popularity and also achieved great results [25, 12, 24], due especially to their ability to aggregate information from neighbor nodes.
However, most of the GCN models focus purely on user-item interactions and do not take into account additional information that will usually be available in both existing datasets and real life.
This additional information could be from a user profile including their age and location, or information about the items such as user-specified tags for attractions or the genres of a movie. There is usually also some contextual information available concerning the interaction taking place, which may be something as simple as a timestamp, or information about the current emotional state of the user.
This side-information and contextual information can be beneficial for prediction performance [10, 21], as well as assist in alleviating certain issues such as data sparsity and cold starts [22]. Using this context information is a sub-genre of RS called context-aware recommender systems (CARS) [18], where the associated context of an interaction is taken into consideration when creating a prediction. This means that the data sparsity problem is especially prevalent in CARS since users may not have interacted with a lot of items in a given context, leading to sparsity in the data available for the collaborative filtering process. In this paper, we

will investigate how GCNs can be extended to generate context-aware recommendations while utilizing this additional information. On top of this, we will examine whether or not utilizing this context can improve recommendations in a regular context-free situation. First of all, we will examine the problem and define some foundational knowledge in Section 2.
In Sections 3 and 4 we propose two models, CSGCN-IS and CSGCN-ADJ, as possible methods to utilizing context and side-information in a GCN-based model. We conduct experiments against a prepared set of research questions in Section 5. Finally, we look at related work and conclude upon the paper in Sections 6 and 7.

2. Preliminaries

This section will present some preliminary knowledge and defintions pertaining to the definition of context and side-information, the components of a GCN, how to represent data, and higher-order connectivity in graphs.

2.1 Defining side-information and context

Datasets used for RS are often very sparse, since not every user will interact with every available item, leading to learning issues. Sparsity issues are exacerbated in CARS due to the addition of contextual dimensions that users only interact with a few of [29]. The use of side-information can help alleviate problems with CF in a sparse situation where the users and items do not have many interactions [24]. While GCN-based RS perform well on simple recommendation tasks that include only user-item interactions [25, 12], combining them with the utilization of context is largely unexplored to the best of our knowledge. However, side-information has been used in some GCN-based RS such as KGAT through the use of knowledge graphs [24].
A common problem when researching CARS is that context and side-information are vaguely defined or have overlapping definitions in various papers. For clarity, the following are the definitions of context and side-information used in this paper:

Context: The context definition used in this paper is identical to the one found in [7], where *context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the applications themselves.* An entity in the case of an RS is either a user or an item to be recommended. The context would thus be information about the conditions at the time of the interaction between the user and the item, meaning it would be associated with the interaction. For example, a Christmas movie may be more likely to be a relevant recommendation during the winter months than in the summer, and a bar may be a better recommendation on a Saturday night than a Tuesday morning.
Throughout this paper, we will refer to context in the three following levels of granularity:

All possible context values, C, e.g.:

$$C = \{spring, \cdots, winter\} \cup \{morning, \cdots, midnight\}$$

Context values for a given interaction, C', e.g.:

$$C' = \{winter, midnight\}$$

A single context value, c, e.g.

$$c = winter$$

Side-information: The side-information definition used is based on the description in [22], where different types of side-information include information such as user networks, user profiles, and item descriptors. For users, this could be information like their age or occupation, which may have an influence on their preferences. For items, it is information about the item such as the genre of a movie, which may be helpful since it can help capture user preferences, such as a user displaying a preference for action movies.

2.2 The components of a GCN

GCNs are variants of convolutional neural networks (CNN) used on graph-based data [16]. GCNs learn a representation of each node in a graph through multiple graph convolution layers. For layer l of the graph convolution, the node representations that were output from the previous layer $l-1$ are used as input. For each layer, the representations of the nodes in the graph are aggregated with the representations of their neighbor nodes. A nonlinear activation function is also traditionally applied at each layer to introduce non-linearity to the model [16]. Representations are propagated with information about their neighbors in each graph convolution layer through three steps: feature propagation, linear transformation, and nonlinear activation. Feature propagation averages features in the local neighborhood of a node, such that the representation of each node becomes an aggregation of its neighbors' features. This is done for each node in the graph. The next step is to use a linear transformation, such as $x_1 = x_0 w_0$ where the embedding x_0 is updated by multiplying it with a weight w_0, thus becoming x_1. Doing this allows the network to find the best fitting set of parameters through the linear transformation by training through optimizing a loss function [8]. Finally, a non-linear activation function such as `ReLU` is applied to the output, allowing it to better fit real-world datasets.

While traditional GCN models contain all three components, several papers [26, 12] suggest that this is largely caused by the history of GCNs. Wu et al. present the idea that while most machine learning algorithms followed a clear path from an initial simple model to a more complex model through the needs of more expressivity, this was not the case of GCNs [26]. Instead, they were proposed in the recent years of neural networks where deep learning had gained high popularity, and as such GCNs were built directly on top of a complex model. To investigate whether this was useful, they present simple graph convolution (SGC), a simplified version of GCNs that could have preceded GCNs if the traditional path had been taken. This led them to remove the linear transformation and nonlinear activation, to reach a simplified linear model. Interestingly, they proved that not only does this not degrade performance of the models, it generally matches or improves both in terms of performance and speed [26].
This signifies that the expressive power of GCNs originates primarily from propagation in the convolution layers, rather than the nonlinear feature extractions we know from traditional GCNs [26].
Having defined GCNs, their applicability for RS tasks is now examined in terms of data and its representation.

2.3 Types of data for RS

Data used for RS can be classified as either implicit or explicit [17]. Explicit data is provided by the user of the RS, such as a user providing a rating of an item. Data provided explicitly by users is commonly used for the score prediction task, where, as with traditional matrix factorization (MF) approaches, a score matrix is reconstructed in order to generate missing ratings for users. Implicit data, on the other hand, is not explicitly provided by the user, but rather collected through their behavior. A user clicking a specific item on a web-shop or deciding to watch a specific movie would be implicit data, defining an interaction between the user and the item. Since explicit data requires explicit user involvement, it is usually more scarce than implicit data. A problem with implicit data, however, is that it does not directly encode whether or not the user had a preference for the given item as is possible with explicit data, only that they interacted with it. A common prediction task for implicit data is top-k recommendation, where a list of k items is produced for the user.

2.4 Data representation

In graph-based CF RS, data is typically represented in the form of a bipartite graph as seen on Figure 1. This bipartite graph consists of two distinct sets of nodes: Users U and items I, and a set of undirected edges E that connect the user and item if they have interacted with each other. This graph can also be represented as an adjacency matrix, which is useful for the implementation of a graph convolutional layer as it allows for the implementation of graph convolution through matrix multiplication between the adjacency matrix and the embeddings of the graph nodes. This is the case since the values being

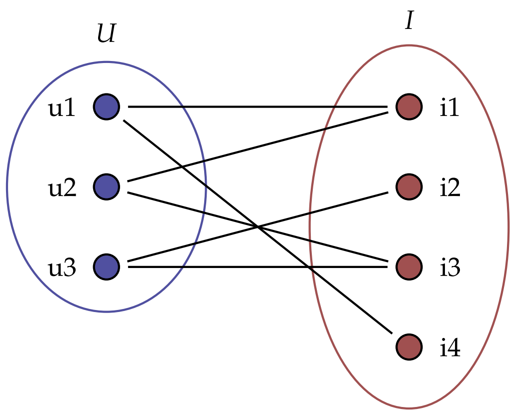


Figure 1: *Example of a bipartite graph.*

multiplied with the representation are from the adjacency matrix that defines the neighbor nodes, thus aggregating the neighborhood.
This could, however, cause issues in terms of exploding or vanishing gradients, leading to scenarios where the method ends up unable to learn. To mitigate this, a degree matrix consisting of the summation of row entries in the diagonal can be employed in order to normalize the entries in the adjacency matrix by the number of neighbors, to ensure the gradients do not grow too large or small. This bipartite graph can be extended to also include nodes that represent side-information, which are connected to the user and item nodes. If side-information is represented as nodes in the graph, they can be included in the adjacency matrix and the graph convolution such that information can be passed from these nodes as well. Higher-order connectivity can be gained by repeatedly doing convolution operations in multiple layers, meaning information is passed from neighboring nodes further away from the central node [26, 16], based on the number of layers. A two layer GCN would, for example, be able to pass information from nodes two steps from the central node, allowing the collaborative signal to be influenced by nodes that are further from the central node.

With this preliminary knowledge in place, we will look at the two proposed models in the following sections.

3. CSGCN-IS

This section describes the first of our two proposed models, Context and Side-information GCN with Item-Splitting (CSGCN-IS), which is a GCN that incorporates side-information in the graph and considers context as an embedding that is used as a modifier for an item.

3.1 Model intuition

Before diving into the details of how CSGCN-IS is modeled, let us examine the intuition behind it. The goal is to construct a model that can generate a top-k list of recommendations for a user. To do this, we start by expressing the available information in a graph. This mainly includes interactions between users and items. However, since most RS datasets are sparse, we want to see if we can alleviate this problem by further connecting user and item nodes, to improve the collaborative signals in the model. To do this, we utilize the fact that most datasets include some side-information about users and items, which can be used for further connections by its addition as nodes in the graph.
Besides side-information, we need to reason about how to express context in the model. In this model, we express context by simplifying it to be an embedding that is used as a modifier to an item to signify whether the item is popular in a given context or not. For example, we may learn that indoor activities are generally more popular on a rainy day. This can be learned by the model through sampling user interactions in various contexts from the ground truth. For the user and context of the positive sampled interaction, we see that they interacted with a given item, which should make the item more likely to be recommended in that context. A negative interaction is also sampled for the same user and context, which should make the item less likely to be recommended for that context through training, further detailed in Section 3.6. With this intuition, we will now detail how this is implemented in the CSGCN-IS model.

3.2 Model architecture

The CSGCN models are based on the observations made in SGC [26] and the LightGCN model [12] about simplifying the models to a linear model. To facilitate the model architecture, each node in the graph is represented by a low-dimensional vector called an embedding. By doing this, the model can learn the nodes' properties and capture them in the embeddings. The model consists of four parts: Input, graph convolution layers, layer combination, and a prediction function.

The input is represented as a quadripartite graph consisting of users, items, user side-information, and item side-information. This graph is used as the input for the graph convolution layers, which serve to capture higher-order connectivities between the nodes in the graph. Each convolutional layer is stacked on top of the previous layer, such that each layer captures information from nodes that are further away from the central node. Having passed through L convolution layers, we are left with an embedding representation of each node for each layer, which is then passed to the layer combination function, taking a weighted sum of each representation to generate a final representation for each node. Finally, these combined representations of users and items are passed to a prediction function along with a context embedding to calculate a score for the given user, item, and context combination as described in Section 3.1.

These predicted scores are used to generate a top-k list of recommended items for a user in the given context.

In the following sections we will further explain how each of these four parts of the model work.

3.3 Adjacency matrix

As described in Section 2, GCNs used for CF recommendation typically employ a bipartite graph structure to represent the user-item interactions. In CSGCN-IS the extended graph now contains users, items and edges as well as the set of user side-information nodes US and the set of item side-information nodes IS. This allows for the incorporation of side-information in the model, whereas context is later included in the prediction function defined in subsection 3.5. This graph extension can be seen on Figure 2. It should be noted that the distinct sets cannot be self-connected, and edges can only exist between adjacent sets. Formally, we let G be a graph consisting of a set of edges E, and vertices V, where $V = US \cup U \cup I \cup IS$.

$$E_{US} \subseteq \{(x, y) |\, x \in US,\, y \in U\}$$

$$E_{U} \subseteq \{(x, y) |\, x \in U,\, y \in US \cup I\}$$

$$E_{I} \subseteq \{(x, y) |\, x \in I,\, y \in U \cup IS\}$$

$$E_{IS} \subseteq \{(x, y) |\, x \in IS,\, y \in I\}$$

$$E = E_{US} \cup E_{U} \cup E_{I} \cup E_{IS}$$

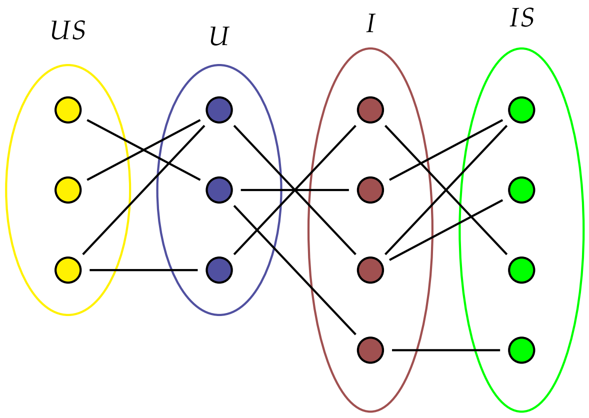


Figure 2: *Example of the quadripartite graph structure used for CSGCN-IS.*

The quadripartite graph can be represented as an adjacency matrix $A \in \mathbb{R}^{|U|\times|I|\times|US|\times|IS|}$. An entry is 1 if the user has interacted with the item, if the item has the given side-information or if a user has the given side-information, and 0 if none of these are true. A consists of the rating sub-matrix R and its transpose R^T, as well as sub-matrices containing US, IS, and their transposes, as seen on Equation (1).

$$\begin{bmatrix} 0 & R & US & 0 \\ R^T & 0 & 0 & IS \\ US^T & 0 & 0 & 0 \\ 0 & IS^T & 0 & 0 \end{bmatrix} \tag{1}$$

Since there are no direct interactions between users and other users, the first quadrant is a zero-matrix of size $|U| \times |U|$ where $|U|$ is the amount of users in the dataset. This goes for all the zero matrices seen on A, making it mostly sparse. Adding the side-information to the adjacency matrix allows us to express the connections between users and items

and their side-information in the graph, allowing it to be used in order to gain higher-order connectivity in the convolution layers.
It is worth noting that, unlike traditional GCNs, CSGCN does not utilize self-connections since the layer combination performed after the convolution layers captures the same effect as self-connections through the 0th layer [12].

3.4 Convolution layers

A major part of the model is the convolution layer where the user and item embeddings are propagated with information about their neighbor nodes' representations by using an aggregation function, which in this case is a normalized sum. Neighbor nodes for users are item nodes and user side-information nodes, and for items the neighbors are user nodes and item side-information nodes, which can be seen on Figure 2. For each additional layer, information is passed from nodes that are further away from a central node, illustrated on Figure 3. The representations at each of the layers are aggregated into a final representation after L layers. From these final representations, a score prediction can be calculated between an item, user and a context which is further described in Section 3.5. The inputs to the layer is the adjacency matrix previously described.

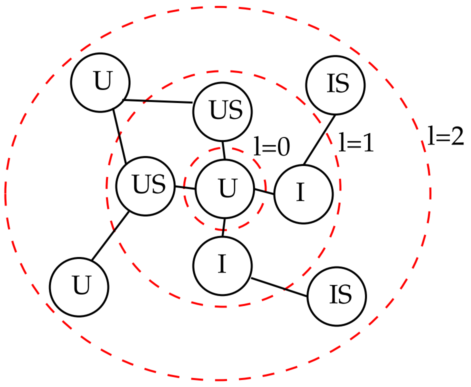


Figure 3: *Information aggregated through layers.*

$$e_i^{(l+1)} = \sum_{in \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_{in}|}} e_{in}^{(l)} \tag{2}$$

$$e_u^{(l+1)} = \sum_{un \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_{un}|}\sqrt{|\mathcal{N}_u|}} e_{un}^{(l)} \tag{3}$$

In Equations (2) and (3) the convolution operation is shown for respectively item and user embeddings, where $\mathcal{N}_u$ are the neighbor nodes for user u, where $\mathcal{N}_u \subseteq US \cup I$, and $\mathcal{N}_i$ are the neighbor nodes for item i where $\mathcal{N}_i \subseteq IS \cup U$.
A user neighbor node is denoted as $un \in \mathcal{N}_u$, and for item neighbors as $in \in \mathcal{N}_i$. N_{un} denotes the set of neighbors for the neighbor un of node u, and N_{in} denotes the set of neighbors for the neighbor in of node i.
This means that when neighbor nodes' representations are being propagated, the normalization is done with both the number of neighbors for the central node and the number of neighbors of the neighbor node that is currently having its representation propagated.

The convolution layer propagates information to user and item nodes' representations by summing the normalized representations of the neighbor nodes. All types of nodes are propagated with information about their neighbors, but since only the user and item nodes' representations are used in the prediction function for our model, their representations are the only ones that are output from the convolution layer.

3.5 Score prediction

To include context in the score prediction, we model each pair of items and context values as learnable embeddings e_{ic}. The use of e_{ic} is heavily inspired by Ricci and Baltrunas' item splitting approach [2], where a context weight for each item in each context is calculated. This means that for the set of items I and the set of possible context values C, there are $|I| \times |C|$ embeddings of item and context value combinations e_{ic}. The objective is to calculate a score between a user and an item in a given context C'. The different embeddings e_{ic} of a given item i and context C' are added to the embedding of i to signify its popularity in the given context. After the addition of the e_{ic} embeddings to the item embedding e_i, the cosine similarity scores of the user embedding e_u and the item-context additions are calculated and summed together to produce a score:

$$\hat{y}_{(u,i,C')} = \sum_{c \in C'} e_u^T (e_i + e_{ic}) \tag{4}$$

As shown in Equation (4), the context embeddings are not part of the convolution layer but are only used in the score prediction, meaning that they are learned directly through their appearance in the score prediction function.

3.6 Training

For the top-k recommendation task, we utilize negative sampling and Bayesian Pairwise Ranking (BPR) loss [20] as the training function, shown in Equation 5.

$$Loss = \sum_{(u,i,j,C') \in D_T} \sigma(-(\hat{y}_{(u,i,C')} - \hat{y}_{(u,j,C')})) + \lambda_\Theta ||\Theta||^2 \tag{5}$$

where Θ denotes the trainable model parameters, which in this case are the 0th layer embeddings for items $e_i^{(0)}$ and users $e_u^{(0)}$, as well as the embeddings for side-information, e_{us} and e_{is}, and item-context embeddings e_{ic}.
λ denotes the regularization coefficient, and $\sigma(\cdot)$ denotes the non-linear activation function, in this case, the `softplus` function.
The regularization term $\lambda_\Theta ||\Theta||^2$ is included to prevent overfitting.
$D_T = \{(u, i, j, C'') | i \in I_{(u,C')} \land j \in I \setminus I_{(u,C')}\}$ denotes the dataset of training tuples, where $I_{(u,C')}$ is the set of observed items that user u has interacted with in context C'. $\hat{y}$ is the score prediction defined in Equation (4) for respectively a positive interaction (u, i, C') between a user u and an item i in context C', and a negative interaction (u, j, C') for the same user in the same context, where j is an item the user has not interacted with.
The positive interaction is an observed interaction between a user and an item, which can be found in the training set. An item that the same user has not interacted with in this given context is randomly sampled as a negative sample, with the assumption that an unobserved interaction is equal to a negative interaction when working with implicit data.

The model parameters are updated by optimizing the BPR loss function in Equation (5). For a specific model parameter, this is done by calculating the gradient of the loss function at each iteration with respect to the model parameter, and then updating the parameter according to the gradient by a step size defined through the learning rate. The gradient is a vector of partial derivatives which defines the slope of the function, where a partial derivative of a function with multiple variables is the derivate with respect to a single variable with the remaining variables held constant. The derivative defines how the output of the function changes based on changes in the input.
The model parameters can then be updated in the negative direction of the slope of the function, the gradient, based on the learning rate in order to approach a local minima. For CSGCN-IS, the loss function is optimized using the *Adam Optimizer* [15]. Optimizer functions are an essential part of machine learning, since they are used to tune the parameters of the neural network to minimize the loss function. The chosen optimizer can impact the training speed and final performance of the model. However, there is no theory that explains how to decide which optimizer to use [6]. Unlike simpler optimizers that maintain a single learning rate throughout training, such as stochastic gradient descent, Adam utilizes an adaptive per-parameter learning rate by estimating the mean and the variance of the gradient and the squared gradient [15]. Additionally, Adam is reliable for calculating gradients in sparse situations, which is an important property for RS, since they often deal with sparse data.

3.7 Implementing CSGCN-IS

To facilitate the implementation of the GCN layer in CSGCN-IS, the adjacency matrix representation of the graph is utilized. The matrix form of CSGCN-IS is similar to the one described in [12], except for the adjacency matrix being modified by including side-information as described in Equation (1). An embedding matrix is introduced which contains each of the embeddings of the nodes in the graph. Let the embedding matrix for the 0th layer be $Emb^{(0)} \in \mathbb{R}^{(|U|+|I|+|US|+|IS|) \times K}$, where K is the embeddings size, then the embedding matrix for the next layer $l+1$ can be obtained by:

$$Emb^{(l+1)} = (D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) Emb^{(l)} \tag{6}$$

where l is the current layer and D is a $(|U| + |I| + |US| + |IS| \times (|U| + |I| + |US| + |IS|)$ degree matrix. Finally, the embeddings from each layer are combined into the final embeddings that are used in the prediction.

$$Emb = \sum_{l=0}^{L} \frac{1}{L} Emb^{(l)} \tag{7}$$

The combination of the layers is seen on Equation (7), and it shows that the final embeddings are an average of the embeddings from each layer.

4. CSGCN-ADJ

This section describes the second proposed model, CSGCN with Context in Adjacency Matrix (CSGCN-ADJ).

4.1 Model intuition

With the second proposed model, context is included in the convolution by adding context nodes to the graph, which will be used to represent interactions between users and items in a given context as a tuple (u, i, C'), implying that a user u has interacted with item i in context C'. The adjacency matrix will now include interactions, which contexts the users have interacted in, and the contexts in which items have been interacted with. This approach is unable to capture which specific context a given interaction occurred in, it only captures whether or not the user or item has had an interaction in the given context. This differs from *CSGCN-IS* since we now model context in a similar fashion to side-information, where each context value has a single embedding instead of having an embedding for each context value and item pair. The intuition is that since context is now modeled similarly to side-information by the addition of nodes, it makes sense to include it in the convolution, since they will be present as neighboring nodes to items and users in the graph. By including nodes for the contexts in the convolution, their representations are propagated with information about their neighbors, being items and users that have interacted in that context, using a normalized sum aggregation. This means that through higher-order connectivity, items and users are propagated with information about other users and items that have interacted in the same contexts. Even though the intuition changes, the model architecture is almost the same as defined in Section 3.2. The difference is that the input graph to the convolution layers now includes context nodes, meaning they are also a part of the neighborhood aggregations.

4.2 Adjacency matrix

Compared to CSGCN-IS, context is now included in the graph as nodes and therefore also included in the adjacency matrix. The graph is similar to the one in Figure 2, but context nodes are inserted between the sets of item and user nodes, and edges connect the nodes that have interacted in the context. This results in the sets of edges E_U and E_I being updated to:

$$E_U \subseteq \{(x, y) | \, x \in U, \, y \in US \cup I \cup C\}$$

$$E_I \subseteq \{(x, y) | \, x \in I, \, y \in U \cup IS \cup C\}$$

A new set of edges is also introduced for the context value nodes in C:

$$E_C \subseteq \{(x, y) | \, x \in U \cup I, \, y \in C\}$$

The new set of edges E_C has directed edges from items or users to context nodes, as well as self-connections for each context node. For the convolution, we want to preserve the context's initial embeddings from the first layer through to the last layer, since we found that this performed better compared to also propagating information about neighbors to the context embeddings. Therefore, the transposed matrices of UC and IC are not inserted into the adjacency matrix. Instead, we add the identity matrix I to the last quadrant such that the context embeddings are only propagated with information about themselves in each convolution. The matrix is seen in Equation (8).

$$\begin{bmatrix} 0 & R & US & 0 & UC \\ R^T & 0 & 0 & IS & IC \\ US^T & 0 & 0 & 0 & 0 \\ 0 & IS^T & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix} \tag{8}$$

4.3 Convolution layers

For the graph convolution layer of CSGCN-ADJ, the context is incorporated into the aggregation of neighbors seen in Equations (9) and (10)

$$e_i^{(l+1)} = \sum_{in \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|}} e_{in}^{(l)} \tag{9}$$

$$e_u^{(l+1)} = \sum_{un \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}} e_{un}^{(l)} \tag{10}$$

With this convolution, only the number of neighbors of the node that information is propagated to is used for normalization. As a result of context being added as nodes in the graph for CSGCN-ADJ, $\mathcal{N}_u$ also contains the contexts that user u has interacted in and $\mathcal{N}_i$ also contains the contexts that item i has interacted in. Because the identity matrix was added to the adjacency matrix under the context columns, the context embeddings retain their initial values throughout the convolution layer.

4.4 Score prediction and training

For the score prediction of CSGCN-ADJ, we draw inspiration from factorization machines and use an FM-like predictor function that considers pair-wise interactions between users, items and context values to provide a recommendation. To do this, we use the prediction function defined in Equation (11):

$$\hat{y}_{(u,i,C')} = e_u^T e_i + \sum_{c \in C'} e_u^T e_c + \sum_{c \in C'} e_i^T e_c \tag{11}$$

Where e_c is the embedding representation of the context value c, e_i is the embedding of the item i and e_u is the embedding of the user u. In this predictor function, we calculate the pairwise interaction between each of the embeddings. The self-interactions are not calculated since they do not contribute to the score prediction. The embeddings used for items and users are the ones output by the convolution layer while the context embeddings are the initial values before convolution, since they are only propagated with information about themselves.
Training of CSGCN-ADJ is similar to that of CSGCN-IS, using the same loss function and the Adam optimizer.

4.5 Implementing CSGCN-ADJ

Let the embedding matrix for the 0th layer be $Emb^{(0)} \in \mathbb{R}^{(|U|+|I|+|US|+|IS|+|C|)\times K}$, where K is the embedding size and C is the set of possible context values. The embedding matrix for the next layer $l+1$ after layer l for CSGCN-ADJ is obtained by:

$$Emb^{(l+1)} = (D^{-\frac{1}{2}}A)Emb^{(l)} \tag{12}$$

where l is the current layer and D is a $(|U|+|I|+|US|+|IS|+|C|) \times (|U|+|I|+|US|+|IS|+|C|)$ degree matrix. We see that for this model, we only normalize on a row-level by multiplying by $D^{-\frac{1}{2}}$ on the left side of the adjacency matrix. This approach is inspired by a combination of the random walk Laplacian $D^{-1}A$ and the symmetrical normalized Laplacian $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ used in CSGCN-IS. We found that for this model, it gave slightly better performance compared to these regular normalization methods, as evidenced by the results seen on Figure 4. While this graph only shows the results for NDCG@20 on the Yelp-NC dataset, the trend is consistent across datasets and metrics.

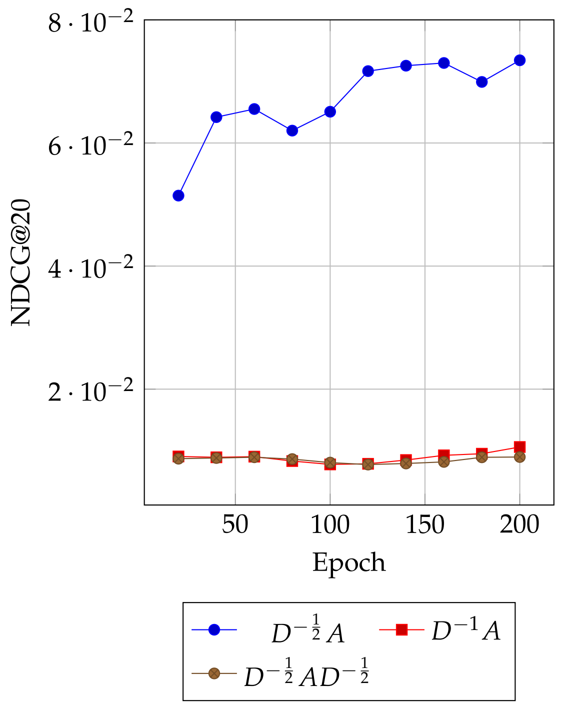


Figure 4: *Effect of various normalizations for CSGCN-ADJ on the Yelp-NC dataset for HR@20.*

The final embedding matrix after L layers is obtained in the same way as CSGCN-IS where we take the mean of the nodes' representations at each layer.

5. Experiments

Experiments are conducted in this section in order to answer the following research questions:

- **RQ1**: How do the CSGCN models compare to state-of-the-art methods for the task of context-aware recommendation list prediction?
- **RQ2**: Can the CSGCN models be used to make non-context specific recommendations?
- **RQ3**: Does side-information improve results for context-aware recommendations?

The section details the experimental settings used, the datasets, the methods, and the metrics employed for comparison.

5.1 Datasets

To test the models on datasets with varying sizes and density, the following datasets were chosen:

Dataset	User #	Item #	Interaction #	Context #	Sideinfo #
ML1m	6,040	3,377	999,416	1	4
Frappe	546	821	85,541	6	1
Yelp-NC	2,274	2,140	130,627	1	4
Yelp-ON	5,185	5,566	332,291	1	4

Table 1: *Statistics of the datasets.*

MovieLens 1M (ML1M)

ML1M is a dataset [9] with 1 million interactions between 6,040 users and 3,377 movies.
We use the following side-information for users:

- Age
- Gender
- Occupation

Side-information for items is *genre*, which specifies that an item has one or more of the 18 genres available in the dataset. For contextual information, the MovieLens dataset provides a timestamp for when the interaction took place.

Frappe

The Frappe dataset [3] contains information about how users have interacted with applications on their smartphones. The only available side-information is for items, indicating whether the application is free or not. However, it contains the following contexts:

- City
- Country
- Weekday
- Time of day
- Whether it is weekend or not
- Weather

Yelp-NC and Yelp-ON

The two Yelp datasets, Yelp-NC and Yelp-ON, are subsets of the public Yelp dataset [28], where we take the subset of items located in respectively North Carolina (NC) and Ontario (ON). Both Yelp datasets have been pruned such that they include only users with at least 10 interactions, to facilitate generating a stratified data split where each user in the test set is also in the training set.
We use the following side-information about users for both datasets:

- Yelping since (Year)
- Number of fans
- Average stars

The only side-information about items is the categories that an item belongs to. For the Yelp-NC dataset, this is one or more of 80 different categories, and 75 categories for Yelp-ON.
The contextual information is a timestamp.

5.2 Context dimension selection

When selecting context dimensions to use from the dataset, some of the context information can benefit from being discretized. This is the case for contexts such as a timestamp, as the information about the interaction is too specific for the model to be able to learn anything valuable from it. This is because there are often not enough interactions for a given context to be able to learn its effect if it is provided in too much detail, as that creates too much sparsity. For timestamps, this problem would occur since each second or minute is modeled differently, meaning the context would only match for the exact same date and second. Therefore, we discretize context that can have too many different values to avoid these sparsity problems. Since most of the datasets include a timestamp as contextual information, we discretize these into the following context dimensions:

- Month
- Day of week
- Hour
- Time of day

Where time of day is a further discretization of the hour dimension, split into 5 intervals:

- Night (0 to 4)
- Early morning (4 to 8)
- Late morning (8 to 12)
- Evening (12 to 16)
- Afternoon (16 to 20)
- Late night (20 to 24)

For the Frappe dataset, we instead make use of the 6 contextual values available in the dataset.

For the side-information, we use the dimensions mentioned in Section 5.1 for both users and items with some discretization. The Yelp datasets provide a "yelping since" dimension, which is a timestamp for the creation of the user. This is discretized to include only the year that the user has been created.
Additionally, the dimension "fans" is discretized into four intervals:

- < 50 fans
- $50 - 100$ fans
- $100 - 500$ fans
- > 500 fans

5.3 Compared methods

In the experiments, the CSGCN models are compared against the following methods for context-specific recommendations:

- Factorization Machine (FM) [19]
- Convolutional Factorization Machine (CFM) [27]
- Neural Factorization Machine (NFM) [11]

FM is a relatively simple model that models all interactions between variables using factorized parameters. It is often used for comparison in context-aware papers, since it is able to predict a score based on information about a given interaction including contextual information. CFM uses a combination of a CNN and an FM, which is similar to CSGCN in that it is capable of producing a context-aware top-k recommendation list. It is included since it is an alternative way to handle contextual information in CNNs. We also compare against NFM, which uses a multi-layer perceptron (MLP) on top of the factorization machine to learn higher-order interaction signals.

To compare the performance of the models in a non-context specific setting, we compare against the following methods:

- Neural Graph Collaborative Filtering (NGCF) [25]
- Light Graph Convolutional Network (LightGCN) [12]
- Knowledge Graph Attention Network (KGAT) [24]
- Bayesian Personalized Ranking (BPR) [20]
- Top Pop

NGCF is a model proposed by Wang et al. as a way to utilize the GCN model proposed in [16] for recommendation purposes. It makes use of all three parts of a GCN described in Section 2, and utilizes BPR loss with negative sampling for ranking purposes. LightGCN was later proposed by the same authors and is mainly focused on performing an ablation study similar to the one done in [26], where the linear transformation and non-linear activation between layers are removed to provide a simpler model, while still presenting competitive results. Both the CSGCN models and LightGCN extend the codebase of NGCF, so they are naturally included in the comparison to show that adding side-information and contextual information hopefully improves the accuracy of these methods.
KGAT is a state-of-the-art graph-based method that utilizes side-information in the form of a knowledge graph. BPR utilizes implicit feedback to generate a personalized top-k list of items using the proposed BPR-optimizer to train the model, providing a sim-

pler method for comparison. Finally, a we include a simple baseline, Top Pop, which simply recommends the k most popular items for all users.

5.4 Parameter settings

To fairly compare the performance of the models, we train all of them by optimizing the BPR loss with Adam. The learning rate is either set to the value presented in the respective papers, or searched between $[0.1, 0.01, 0.001, 0.0001]$ if a default is not available. The batch size is set as 1024 for all datasets except Frappe, where it is set to 512 due to its size. The embedding size is set to 64, and for the models using regularization terms, this is searched between $[0.01, 0.001, 0.0001]$ if a default value is not provided. For CFM and NFM we follow the approach presented in the original papers and feed them with weights from a pretrained FM model which has run for 500 epochs with the settings presented in the CFM paper. All methods are run for a maximum of 1000 epochs with an early stopping mechanism triggered when the HR@20 or Recall@20 has not improved for 5 evaluations (100 consecutive steps), except for CFM which is run for 300 epochs based on the default values.

5.5 Evaluating the models

The objectives of the models in the experiments are to produce top-k lists of items for a user. Different methods for evaluating the performances of the models were considered, specifically the fold-out and leave-one-out methods. These methods interact with the context in different ways. For the fold-out method, the datasets are split into a training set and a test set, where the model is trained on the training set and evaluated on the test set. This split is generated such that each user in the test set is also found in the training set. Using this stratified split method, we decided to use a split of 80% training and 20% test split. Because of this split, a single user can have multiple ground truth entries across multiple contexts.
A problem to consider with fold-out for context-aware evaluation is that when the context is part of the input to the model, a score must be calculated for each combination of item and context due to the multiple ground truth contexts, which can be computationally expensive. A single context cannot be guaranteed for the user for all ground truths, and thus every context needs to be evaluated.

For the leave-one-out method, the newest interaction for each user is removed from the training set and used as the test set, as per [27, 20]. This means that when a top-k list is produced for a user, only a single item amongst the recommended items can be relevant, and thus there is only one context within which the ground truth had occurred. This approach is largely inspired by sequential recommendation systems, where the newest interaction from each user is left out and the next item in the sequence is predicted based on the sequence of the previous interactions by the user [1]. This approach limits the number of different metrics that can be used for this evaluation method due to the single interaction available for testing.

For comparisons we conduct an experiment using fold-out for models that produce a top-k recommendation list and conduct a separate leave-one-out evaluation for models that produce a context-aware top-k recommendation list. The metrics used for the fold-out evaluation are Precision, Recall, and Normalized Discounted Cumulative Gain (NDCG), while the leave-one-out experiments report Hit Rate (HR) and Mean Reciprocal Rank (MRR).

Evaluation metrics explained

The top-k recommendation problem is the task of recommending a list $L(u)$ to an active user u containing k items. Evaluating the quality of a method can be done by splitting the set of items I into a training set I_{train} and a test set I_{test} for each user.
Let $T(u)$ be the the set of items that the user has interacted with in the test set. Precision can be calculated according to Equation (13), and recall according to Equation (14). Precision defines the proportion of relevant items among the predicted items, while recall is the proportion of items that were correctly predicted according to the ground truth.

$$Precision@N(L) = \frac{1}{|U|} \sum_{u \in U} \frac{|L(u) \cap T(u)|}{|L(u)|} \tag{13}$$

$$Recall@N(L) = \frac{1}{|U|} \sum_{u \in U} \frac{|L(u) \cap T(u)|}{|T(u)|} \quad (14)$$

Another metric to be used is NDCG [14]. Assuming each user u has a gain g_{ui} from being recommended item i, the average DCG for a list of J items is defined in Equation (15), where i_j is the item at position j in the list, and the logarithm base b is 2 to ensure all positions are discounted. This metric rewards lists that frontload relevant items.

$$DCG@N = \frac{1}{|U|} \sum_{u=1}^{|U|} \sum_{j=1}^{|J|} \frac{g_{ui_j}}{log_b(j+1)} \quad (15)$$

NDCG is defined in Equation (16), where $iDCG$ is the ideal DCG, which is defined by sorting the recommended items such that the relevant items appear at the start of the list, resulting in the largest possible DCG value.

$$NDCG@N = \frac{DCG}{iDCG} \quad (16)$$

MRR [23] is employed for the leave-one-out evaluation, defined in Equation (17). For each user $u \in U$, MRR is calculated by finding the rank r_u of the first relevant recommendation for each list, and then taking the mean of these ranks. For example, if three items are recommended to a user and the third recommendation is relevant, a rank of $1/3$ is attained for this user. If the recommendation for another user contains a relevant item as the first item of the list, a score of $1/1$ is attained. With this, the final MRR@3 score is $(1/3 + 1/1)/2 = 0.66$.

$$MRR@N = \frac{1}{|U|} \sum_{u \in U} \frac{1}{r_u} \quad (17)$$

The final metric used is HR. If a relevant item for a user appears in the top-k list, the $HR(u)$ for that user is 1, otherwise it is 0. The final HR score is calculated by taking the mean of all users' individual HR scores, as seen in Equation (18).

$$HR@N = \frac{1}{|U|} \sum_{u \in U} HR(u) \quad (18)$$

5.6 Performance Comparison with Context (RQ1)

For the first research question we want to examine how the CSGCN models compare to state-of-the-art methods within context-aware recommendation list prediction. To investigate this, the leave-one-out method presented in Section 5.5 is used, such that a single interaction is left out for each user, and a list is produced based on the context that this interaction took place in. For the tables investing results in all of the section pertaining to the research questions, Sections 5.6 to 5.8, we use the convention of highlighting the best performing method through bold text, and highlighting the second best performing method that is not either of the CSGCN methods through underlined text. The results for this experiment can be seen on Table 2. It is easily seen that both CSGCN-IS and CSGCN-ADJ outperform the various factorization machines on most datasets. The exception is Frappe, a dataset that is fairly unbalanced in terms of which items have been interacted with. The twenty most popular items have between 640 and 6,634 interactions, which is quite a large number for a dataset only containing 85,541 interactions, meaning that almost every 13th interaction will involve the most popular item, whereas all items have a mean of 104 interactions.

From this, we can assume that the CSGCN-IS and CSGCN-ADJ methods work best on fairly balanced datasets, whereas factorization machines seem to have an advantage on datasets with highly popular items.

Another interesting observation from the data is that even though CFM is run on pretrained FM data as described in their paper, it generally performs worse than the traditional FM. Additionally, the runtime of CFM is significantly higher than any of the other methods. Running 300 epochs on an NVIDIA DGX-2 cloud takes several days, while CSGCN-ADJ can perform the same amount of epochs in just a matter of hours. Due to the sheer amount of time required to fine-tune the hyperparameters, CFM has been run only once with the settings presented in the original paper.
We are not able to fairly compare with the results presented in the CFM paper [27], since they perform a different split of the data to perform their leave-one-out evaluation. According to their paper, they take the latest interaction of each user - but

Dataset	Metric	FM	NFM	CFM	CSGCN-IS	CSGCN-ADJ	Impr.
Yelp-NC	HR@20	0.0075	0.0142	0.0070	0.0625	**0.0761**	435.92%
	HR@50	0.0198	0.0240	0.0163	0.1143	**0.1464**	510.00%
	MRR@20	0.0022	0.0030	0.0037	0.0127	**0.0158**	327.03%
	MRR@50	0.0025	0.0033	0.0040	0.0143	**0.0179**	347.5%
Frappe	HR@20	**0.6590**	0.3516	0.3242	0.0458	0.0604	-90.83%
	HR@50	**0.8004**	0.4945	0.4560	0.1062	0.1245	-84.45%
	MRR@20	**0.2895**	0.1117	0.1032	0.0128	0.0184	-1,473.37%
	MRR@50	**0.2942**	0.1166	0.1074	0.0146	0.0204	-1,342.16%
Yelp-ON	HR@20	0.0083	0.0085	0.0073	0.0399	**0.0534**	528.24%
	HR@50	0.0133	0.0152	0.0133	0.0714	**0.0997**	251.32%
	MRR@20	0.0019	0.0021	0.0019	0.0091	**0.0123**	485.71%
	MRR@50	0.0020	0.0023	0.0021	0.0101	**0.0137**	495.65%
ML1M	HR@20	0.0874	0.0937	0.0974	0.1402	**0.1652**	76.31%
	HR@50	0.1856	0.2005	0.1964	0.2603	**0.2982**	48.73%
	MRR@20	0.0199	0.0209	0.0203	0.0336	**0.0372**	77.99%
	MRR@50	0.0229	0.0242	0.0234	0.0373	**0.0413**	70.66%

Table 2: *Results for context-specific recommendations.*

looking at the actual datasets they provide, their test datasets are significantly larger than the number of users, meaning that their evaluation must be different.

In conclusion, the performance of the CSGCN models depends on the dataset characteristics, and the models perform significantly worse than simpler models on the Frappe dataset.

5.7 Performance Comparison without Context (RQ2)

Since the CSGCN models use context to generate a top-*k* list in each context for each user, we argue that this context information can be useful even for non-context-specific settings. Imagine that your system is recommending businesses to users, and you have a business that is only open on Christmas. On Christmas night, millions upon millions of users interact with this item, but it is never interacted with in any other context. In a regular RS this would be considered a popular item, since a lot of users have interacted with it. However, looking at the context, we are able to infer that it is only in this exact context it is popular. This leads to the item having a high likelihood of being recommended in exactly one context, but a low likelihood in any other. With this in mind, we will try to answer RQ2 about whether the models can be used to make non-context specific recommendations through aggregations by investigating various types of aggregations of the scores as well as through a performance comparison with several methods. The simplest solution to aggregate scores would be to take the highest score for an item, regardless of the context that it was recommended for. However, this may prove problematic, since an item may score highly in a single context but poorly in any other context, as in the example before.

Instead, we propose to take an average score across all contexts. With this, the item that is only interacted with in exactly one context has its score lowered by a factor of the number of contexts observed.

The results for the non-context-specific recommendation task can be seen in Table 3. Generally, we see that both CSGCN methods are among the best performers across all metrics on the datasets, with CSGCN-ADJ outperforming CSGCN-IS on all metrics. The main exception is on the Frappe dataset, where BPR and KGAT perform significantly better. We suspect that this is due to the density and small

Dataset	Metric	LightGCN	KGAT	BPR	NGCF	TopPop	CSGCN-IS	CSGCN-ADJ	Impr.
Yelp-NC	Recall@20	0.1069	0.1064	0.0971	0.0923	0.0625	0.1115	**0.1143**	6.92%
	Recall@50	0.1959	0.1935	0.1802	0.1748	0.1226	0.2020	**0.2058**	5.05%
	Precision@20	0.0540	0.0513	0.0482	0.0458	0.0308	0.0547	**0.0575**	6.48%
	Precision@50	0.0402	0.0388	0.0362	0.0355	0.0247	0.0410	**0.0425**	5.72%
	NDCG@20	0.0953	0.0843	0.0765	0.0786	0.0489	0.0995	**0.1020**	7.03%
	NDCG@50	0.1265	0.0994	0.0907	0.1079	0.0588	0.1315	**0.1341**	6.01%
	F1@20	0.0717	0.0692	0.0644	0.0612	0.0413	0.0734	**0.0765**	6.69%
	F1@50	0.0666	0.0646	0.0603	0.0589	0.0411	0.0682	**0.0704**	5.71%
Frappe	Recall@20	0.1101	0.1279	0.1404	0.1029	**0.3787**	0.1054	0.1129	-70.19%
	Recall@50	0.1414	0.1700	0.1762	0.1340	**0.5175**	0.1376	0.1417	-72.62%
	Precision@20	0.0553	0.0495	0.0591	0.0501	**0.1687**	0.0506	0.0576	-65.86%
	Precision@50	0.0306	0.0289	0.0311	0.0284	**0.0968**	0.0291	0.0304	-68.60%
	NDCG@20	0.1141	0.0978	0.1340	0.0986	**0.3695**	0.0997	0.1207	-67.33%
	NDCG@50	0.1203	0.1085	0.1377	0.1057	**0.3775**	0.1073	0.1245	-67.02%
	F1@20	0.0736	0.0714	0.0832	0.0674	**0.2335**	0.0671	0.0763	-67.32%
	F1@50	0.0503	0.0495	0.0529	0.0468	**0.1632**	0.0480	0.0501	-69.30%
Yelp-ON	Recall@20	0.0748	0.0700	0.0658	0.0603	0.0360	0.0751	**0.0809**	8.16%
	Recall@50	0.1354	0.1321	0.1227	0.1157	0.0681	0.1378	**0.1477**	9.08%
	Precision@20	0.0425	0.0395	0.0380	0.0342	0.0200	0.0432	**0.0456**	7.29%
	Precision@50	0.0311	0.0297	0.0285	0.0267	0.0156	0.0316	**0.0335**	7.72%
	NDCG@20	0.0699	0.0610	0.0570	0.0554	0.0303	0.0713	**0.0769**	10.01%
	NDCG@50	0.0909	0.0703	0.0656	0.0755	0.0356	0.0928	**0.1000**	10.01%
	F1@20	0.0542	0.0505	0.0481	0.0436	0.0258	0.0549	**0.0583**	7.56%
	F1@50	0.0506	0.0485	0.0463	0.0433	0.0255	0.0514	**0.0547**	8.10%
ML1M	Recall@20	0.2481	0.2465	0.2642	0.2291	0.0827	0.2550	**0.2675**	1.25%
	Recall@50	0.4144	0.4125	0.4330	0.3874	0.1633	0.4185	**0.4351**	0.48%
	Precision@20	0.2885	0.2874	**0.3015**	0.2663	0.0864	0.2923	0.3013	-0.07%
	Precision@50	0.2110	0.2099	**0.2189**	0.1965	0.0734	0.2125	0.2180	-0.41%
	NDCG@20	0.3754	0.3971	**0.4184**	0.3443	0.1044	0.3822	0.3971	-5.09%
	NDCG@50	0.3966	0.3874	0.4087	0.3662	0.1144	0.4027	**0.4178**	2.23%
	F1@20	0.2668	0.2654	0.2816	0.2463	0.0845	0.2724	**0.2834**	0.64%
	F1@50	0.2797	0.2782	**0.2907**	0.2608	0.1013	0.2819	0.2905	0.07%

Table 3: *Results for the non-context-specific experiment.*

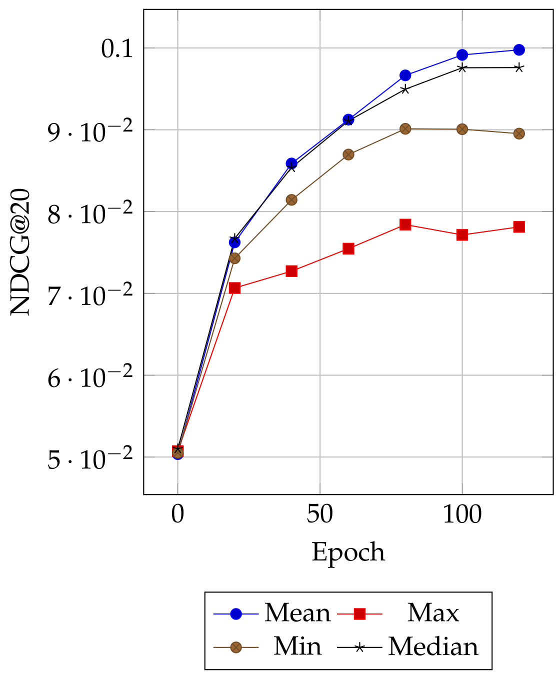


Figure 5: *Effect of aggregation functions for CSGCN-IS on the Yelp-NC dataset for NDCG@20.*

size of the dataset. Frappe has a density of 18.9%, compared to the others that range between 1.1% and 4.8%, allowing for simpler methods like BPR to perform better relative to other datasets, whereas complex methods like NGCF struggle to learn properly. Even more interesting in the Frappe dataset is the performance of TopPop, which outperforms all other methods on all metrics. This is most likely caused by the dataset including some very popular items as described in Section 5.6. All of these results are based on using the mean aggregator, but for the sake of completeness, let us compare the performance for the Yelp-ON dataset across various aggregators of context on CSGCN-IS.

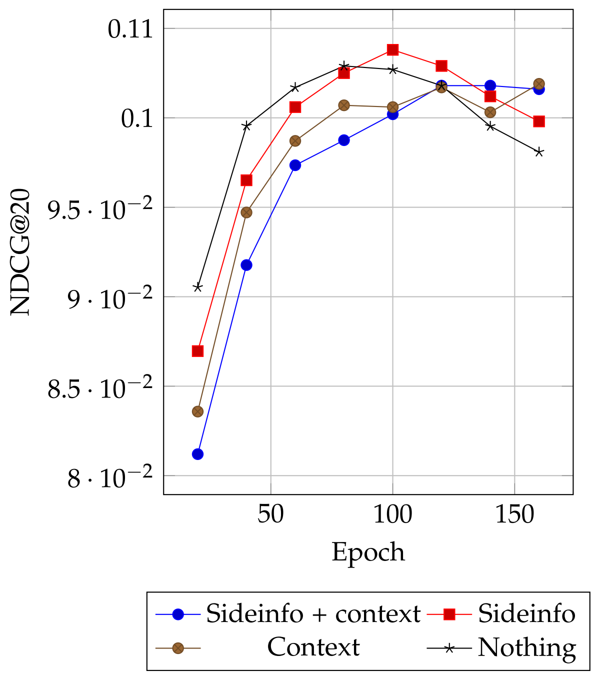
Figure 5 shows how the choice of aggregation affects the CSGCN-IS model. This shows that using a mean aggregation function provides the best results, followed by the median. The simple solution of taking the lowest or highest score across the contexts performs the worst.

This supports the previous assumption that while a single item might be more likely to be recommended in one context, it does not necessarily mean that it is a good recommendation overall, which is better captured through the mean or median aggregations. While the results are only shown for NDCG@20 for this dataset, they have proven consistent across all the metrics.

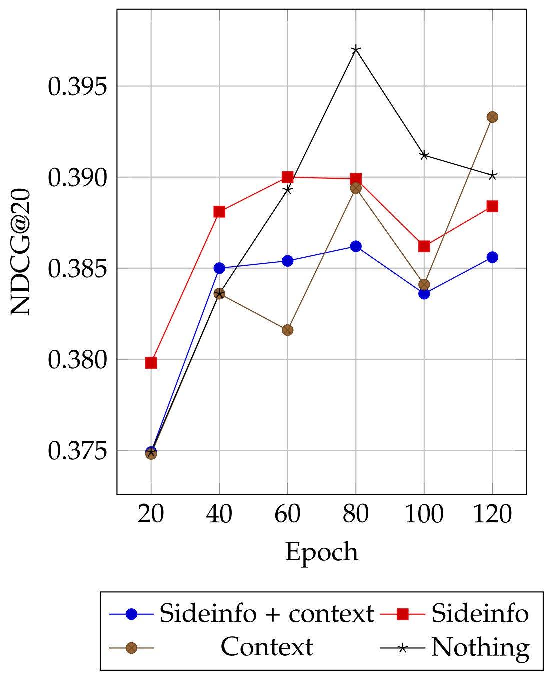
Ablation study for non-contextual recommendation

To test whether context and side-information actually increase performance in a non-context specific setting, CSGCN-ADJ is evaluated using different inputs for the datasets Yelp-NC and ML1M and the performance is measured with NDCG@20. First of all, CSGCN-ADJ is evaluated with both side-information and context. Then side-information is removed, followed by removing context, and finally both context and side-information are removed. Figure 6a shows the performance on NDCG@20 of CSGCN-ADJ with different input. It shows that the more input the model receives, the longer it takes to converge to its best performance. Overall, the performance does not change drastically with the different types of input. Having only side-information as input achieves the best performance, but it is only marginally better than not having side-information and context at all. Having both context and side-information actually degrades performance slightly, compared to just having side-information as the input, and having just context as input improves performance slightly. To conclude from the study on the Yelp-NC dataset, the usage of context is not able to provide better recommendations in a non-context-specific setting. Side-information, however, performs the best on NDCG@20 on this dataset, which suggests that side-information in CSGCN-ADJ is able to slightly improve results compared to having no extra input.

On Figure 6b the NDGC@20 results of ML1M are shown for CSGCN-ADJ with different types of input. For this dataset, the model that uses neither the context nor side-information for the input is actually the best performing version. Side-information and context are not able to improve the performance of CSGCN-ADJ on ML1M in a non-context-specific setting. Since both Figures 6a and 6b show that the model is worse with context included, it can be concluded that context does



(a) *The performance on NDCG@20 of CSGCN-ADJ on the Yelp-NC dataset with different types of input.*



(b) *The performance on NDCG@20 of CSGCN-ADJ on the ML1M dataset with different types of input.*

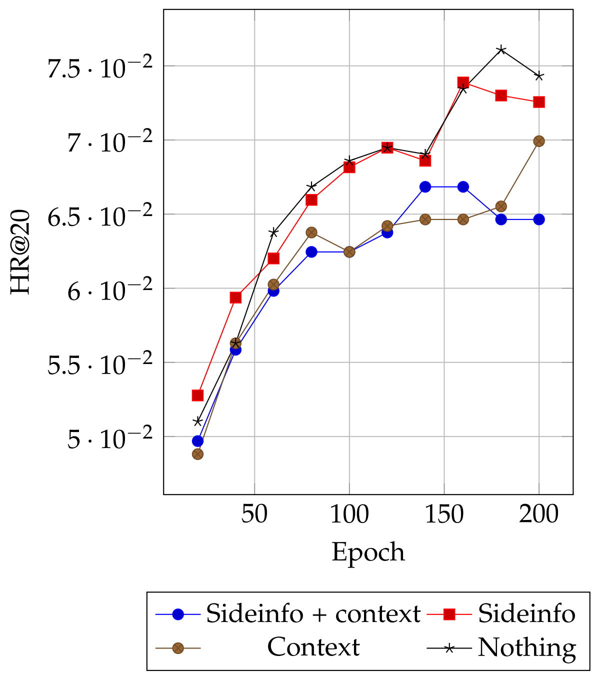
Figure 6: *Ablation study for the input of CSGCN-ADJ on Yelp-NC and ML1M.*

not increase the performance of CSGCN-ADJ in a non-context specific setting. Side-information, however, was able to increase performance of the model in a non-context specific setting on Yelp-NC, but for ML1M, it only makes the results worse compared to having no extra input.

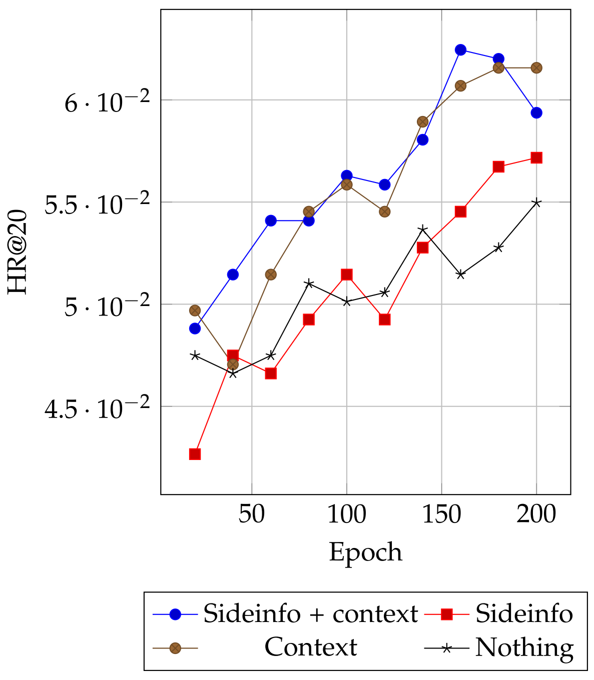
5.8 Ablation study of model extensions (RQ3)

To investigate whether side-information improves the results for context-aware recommendation, we performed an ablation study on the Yelp-NC dataset. In the study, the CSGCN models were evaluated with and without side-information to test the effect it has on the performance. Additionally, the experiment was conducted on the context input to examine how much it affects the performance in a context-specific setting. Figure 7a shows the HR@20 for CSGCN-ADJ on Yelp-NC with different combinations of input. The results show that when side-information is included in the input for CSGCN-ADJ it does not influence the performance much compared to excluding it. Another interesting thing to note is how the model performs worse when context is included in the input, even though it is a context-specific setting. This could be because the CSGCN-ADJ model does not actually model context in a way that fits the definition of context defined in Section 2.1. In CSGCN-ADJ, context can instead be viewed as a type of side-information since it is modelled in a similar way. Each user and item node is connected with the context value nodes of the contexts they have interacted in, just as they are connected to the nodes of their side-information values.

Figure 7b shows HR@20 for CSGCN-IS on Yelp-NC with different inputs. For CSGCN-IS, there is a slight increase in performance when side-information is included, compared to excluding it. This shows that CSGCN-IS is able to use the



(a) *The performance on HR@20 of CSGCN-ADJ on the Yelp-NC dataset with different types of input.*



(b) *The performance on HR@20 of CSGCN-IS on the Yelp-NC dataset with different types of input.*

Figure 7: *Ablation study for the input of CSGCN-ADJ and CSGCN-IS on Yelp-NC in the context-specific setting.*

side-information to better connect user and item nodes, increasing the collaborative signal. We also see that for this model, including context in the input does make a significant different for the performance in a context-specific setting. This is because context is modelled with finer granularity than in CSGCN-ADJ, meaning that each item and context value combination has an embedding. The result of this finer granularity is that context can have different effects on the various items, resulting in more accurate context-specific recommendations.

In general, side-information can have a slight effect on the results depending on the type of model. However, the context seems to have a lot larger effect on the performance of the models, as would be expected for context-aware recommendation. Additionally, it may be worth noting that the context used for these experiments is mainly based on discretized values from timestamps, which might not be a useful contextual dimension due to time zone differences. We do not see any overwhelming evidence that side-information is able to improve context-aware recommendations.

6. Related work

This paper draws inspiration from two primary sources: Graph Convolution Networks (GCN) and Factorization Machines (FM).
This section of the paper will briefly present some methods belonging to those categories and how they relate to our work.

6.1 GCN-based methods

Neural Graph Collaborative Filtering (NGCF) [25] employs a graph neural network to learn embeddings of users and items through an integration of a bipartite graph structure. This structure captures the collaborative signal through high-order connectivity by stacking multiple embedding propagation layers. LightGCN [12] argues that the reasons for the performance of GCN for recommendation purposes are not well understood, and that previous models such as NGCF incorporate unnecessary complexity. LightGCN thus proposes to remove feature transformation and nonlinear activation from NGCF, finding that they contribute little to the performance. LightGCN learns user and item embeddings by linearly propagating them on the bipartite graph structure and uses the weighted sum of the embeddings as the final embedding used for score prediction. Knowledge Graph Attention Network (KGAT) [24] extends the NGCF model by incorporating side-information through a hybrid graph structure of a knowledge graph containing side-information and a user-item bipartite graph, meaning attributes on items can be propagated as nodes, and be used to refine embeddings.

6.2 Factorization Machines

FMs [19] model second-order feature interactions by calculating the inner product. One of the advantages of using FMs is that computations can be done in linear time, while it can also be applied to a variety of prediction tasks such as regression, binary classification and ranking [19]. Convolutional Factorization Machine (CFM) [27] extends FM to the domain of CARS through modeling second-order interactions with an outer product to capture correlations between embeddings, and applying convolution to learn high-order interaction signals. Like CFM, Neural Factorization Machine (NFM) [11] extends the FM model to combine the linearity of FMs with the non-linearity in neural networks. The model extends FMs by adding a number of hidden layers between the bi-interaction layer and the prediction function, such that it should be able to capture both second-order interactions like regular FMs, but also higher-order feature interactions.

7. Conclusion

In this paper, we have proposed two ways to include context and side-information in graph convolution neural networks. Through a series of experiments, we have attempted to answer the following research questions:

- **RQ1**: How do the CSGCN models compare to state-of-the-art methods for the task of context-aware recommendation list prediction?
- **RQ2**: Can the CSGCN models be used to make non-context specific recommendations?
- **RQ3**: Does side-information improve results for context-aware recommendations?

While there are not many competing methods available for comparison in context-specific recommendation scenarios, we have shown that compared to both traditional FMs and deep learning versions of those, the CSGCN models outperform them across three of four real world datasets, ranging from an improvement of 70.66% to 510.0% using a leave-one-out evaluation approach. On the Frappe dataset the CSGCN models are outperformed by 84.45% to $1,473.37\%$. The large fluctuations in improvements and decreases are likely caused by the evaluation methodology where the models attempt to predict a single relevant item amongst every item in the dataset, as well as the small size of the Frappe dataset causing learning difficulties.

To answer whether the models could be generalized to make non-context specific recommendations, we have shown how the context can be handled using a simple mean aggregator and used in a general recommendation scenario with competitive results compared to state-of-the-art methods. The CSGCN models improve performance in a range of 0.07% to 10.01%, while being outperformed on Frappe once again and on the precision metric for ML1M, in a range of 0.04% to 72.62%. The large decrease on Frappe is caused by the data split, which inadvertently facilitated the simple baseline Top Pop. Some items in Frappe are so popular that almost every 13th interaction in the dataset is with these items. While this is a perfect scenario for TopPop, it proved to be a challenge for more advanced methods such as the CSGCN models.
Finally, an ablation study was performed to see how much side-information improves performance for context-aware recommendations. Through these experiments, we see that adding side-information does increase connectivity in the input graph, but it does not consequently improve or worsen recommendations.
In summary, the CSGCN models show im-

provements over compared methods in both context-specific and non-context specific situations across most of the datasets used, and serve as a base for further research on using GCNs for context-specific recommendations.

7.1 Future work

The most prominent future work is further contemplation on how to express context in a way that is compatible with GCN models. Our proposal is to look into ways to represent context as edge information in a feasible way. While writing this paper, we attempted to model it in a way such that there was an embedding for each $(user, item, context)$ tuple. However, this quickly proved infeasible since the number of embeddings explodes with just a few contexts available.

The CSGCN models make use of negative sampling for calculating loss. While this is a regular way to handle BPR loss, it is known to be biased [5, 4] and sensitive to the number of negative samples [13]. Fluctuations in the sampled negative item can cause difficulties in converging, and even sampling more negative items can increase performance.
Chen et al. [5] propose to learn FM models without negative sampling to improve ranking performance for context-aware recommendation, and increase stability due to considering all samples in each parameter update. However, using negative sampling can be necessary since using non-sampling not only requires the method to look at each observed interaction from the dataset, but also to consider each non-observed interaction as a negative data point. Because of this consideration, CSGCN-IS uses negative sampling for training, and, due to employing BPR loss, only one negative sample is used. For future work on the methods, it may be interesting to look into other ways to calculate the loss that are less reliant on hitting good samples.

Additionally, throughout the experiments it was shown that not all contexts are equal in terms of importance. Due to this, it could be beneficial to include an attention mechanism that is able to learn the importance of each context or context combination for the convolution layer. This kind of attention idea could also be applied to layers. More noise will be introduced to the convolutions as the amount of layers in a GCN is increased. To counter this, it may be beneficial to look into more advanced weighting mechanisms for layers, such that nodes further away from the root node are less influential in the layer combination operation.

Finally, both CSGCN methods are based on the simplified models of GCNs which are gaining popularity. For this paper, we have performed limited experimentation on extending this with various components from traditional GCNs, such as re-adding linear transformation and non-linear activation, without success. In recent months, several new papers on GCNs have been published which present new extensions to GCNs. These extensions include new layer aggregation functions and various implementations of knowledge graphs, which may be of interest to improve the performance of the context-aware models presented in this paper.

References

[1] Charu C Aggarwal et al. *Recommender systems*. Vol. 1. Springer, 2016.

[2] Linas Baltrunas and Francesco Ricci. "Context-Based Splitting of Item Ratings in Collaborative Filtering". In: *Proceedings of the Third ACM Conference on Recommender Systems*. RecSys '09. New York, New York, USA: Association for Computing Machinery, 2009, 245–248. ISBN: 9781605584355. DOI: `10.1145/1639714.1639759`. URL: `https://doi-org.zorac.aub.aau.dk/10.1145/1639714.1639759`.

[3] Linas Baltrunas et al. *Frappe: Understanding the Usage and Perception of Mobile App Recommendations In-The-Wild*. 2015. arXiv: `1505.03014 [cs.IR]`.

[4] Chong Chen et al. "An Efficient Adaptive Transfer Neural Network for Social-Aware Recommendation". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR'19. Paris, France: Association for Computing Machinery, 2019, 225–234. ISBN: 9781450361729.

DOI: 10.1145/3331184.3331192. URL: https://doi.org/10.1145/3331184.3331192.

[5] Chong Chen et al. "Efficient Non-Sampling Factorization Machines for Optimal Context-Aware Recommendation". In: *Proceedings of The Web Conference 2020*. WWW '20. Taipei, Taiwan: Association for Computing Machinery, 2020, 2400–2410. ISBN: 9781450370233. DOI: 10.1145/3366423.3380303. URL: https://doi.org/10.1145/3366423.3380303.

[6] Dami Choi et al. "On Empirical Comparisons of Optimizers for Deep Learning". In: *CoRR* abs/1910.05446 (2019). arXiv: 1910.05446. URL: http://arxiv.org/abs/1910.05446.

[7] Anind K. Dey. "Understanding and Using Context". In: *Personal Ubiquitous Comput.* 5.1 (Jan. 2001), 4–7. ISSN: 1617-4909. DOI: 10.1007/s007790170019. URL: https://doi.org/10.1007/s007790170019.

[8] Jiuxiang Gu et al. *Recent Advances in Convolutional Neural Networks*. 2017. arXiv: 1512.07108 [cs.CV].

[9] F. Maxwell Harper and Joseph A. Konstan. "The MovieLens Datasets: History and Context". In: *ACM Trans. Interact. Intell. Syst.* 5.4 (Dec. 2015). ISSN: 2160-6455. DOI: 10.1145/2827872. URL: https://doi.org/10.1145/2827872.

[10] Khalid Haruna et al. "Context-Aware Recommender System: A Review of Recent Developmental Process and Future Research Direction". In: *Applied Sciences* 7 (Dec. 2017), p. 1211. DOI: 10.3390/app7121211.

[11] Xiangnan He and Tat-Seng Chua. "Neural Factorization Machines for Sparse Predictive Analytics". In: *CoRR* abs/1708.05027 (2017). arXiv: 1708.05027. URL: http://arxiv.org/abs/1708.05027.

[12] Xiangnan He et al. *LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation*. 2020. arXiv: 2002.02126 [cs.IR].

[13] Xiangnan He et al. *Neural Collaborative Filtering*. 2017. arXiv: 1708.05031 [cs.IR].

[14] Kalervo Järvelin and Jaana Kekäläinen. "Cumulated Gain-Based Evaluation of IR Techniques". In: *ACM Trans. Inf. Syst.* 20.4 (Oct. 2002), 422–446. ISSN: 1046-8188. DOI: 10.1145/582415.582418. URL: https://doi-org.zorac.aub.aau.dk/10.1145/582415.582418.

[15] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[16] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG].

[17] Douglas W Oard, Jinmook Kim, et al. "Implicit feedback for recommender systems". In: *Proceedings of the AAAI workshop on recommender systems*. Vol. 83. WoUongong. 1998.

[18] Shaina Raza and Chen Ding. "Progress in context-aware recommender systems - An overview". In: *Comput. Sci. Rev.* 31 (2019), pp. 84–97.

[19] S. Rendle. "Factorization Machines". In: *2010 IEEE International Conference on Data Mining*. 2010, pp. 995–1000. DOI: 10.1109/ICDM.2010.127.

[20] Steffen Rendle et al. "BPR: Bayesian Personalized Ranking from Implicit Feedback". In: *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. UAI '09. Montreal, Quebec, Canada: AUAI Press, 2009, 452–461. ISBN: 9780974903958.

[21] Timo Schreiner, Alexandra Rese, and Daniel Baier. "Success Factors for Recommender Systems From a Customers' Perspective". In: 6 (Oct. 2020). DOI: 10.5445/KSP/1000098012/02.

[22] Zhu Sun et al. "Research commentary on recommendations with side information: A survey and research directions". In: *Electronic Commerce Research and Applications* 37 (2019), p. 100879. ISSN: 1567-4223. DOI: 10.1016/j.elerap.2019.100879. URL: http://dx.doi.org/10.1016/j.elerap.2019.100879.

[23] Ellen Voorhees. "The TREC-8 question answering track report". In: (Nov. 2000).

[24] Xiang Wang et al. "KGAT: Knowledge Graph Attention Network for Recommendation". In: July 2019, pp. 950–958. ISBN: 978-1-4503-6201-6. DOI: 10.1145/3292500.3330989.

[25] Xiang Wang et al. "Neural Graph Collaborative Filtering". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (2019). DOI: 10.1145/3331184.3331267. URL: http://dx.doi.org/10.1145/3331184.3331267.

[26] Felix Wu et al. *Simplifying Graph Convolutional Networks*. 2019. arXiv: 1902.07153 [cs.LG].

[27] Xin Xin et al. "CFM: Convolutional Factorization Machines for Context-Aware Recommendation". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 3926–3932. DOI: 10.24963/ijcai.2019/545. URL: https://doi.org/10.24963/ijcai.2019/545.

[28] *Yelp Open Dataset*. https://www.yelp.com/dataset. Accessed: 2021-05-26.

[29] Penghua Yu, Lanfen Lin, and Jing Wang. "A novel framework to alleviate the sparsity problem in context-aware recommender systems". In: *New Review of Hypermedia and Multimedia* 23.2 (2017), pp. 141–158. DOI: 10.1080/13614568.2016.1152319.