# GATMC: Graph Attention Matrix Completion

Derek Xu
*Computer Science Department*
*University of California, Los Angeles*
derekqxu@ucla.edu

Sicheng Jia
*Computer Science Department*
*University of California, Los Angeles*
jsicheng@cs.ucla.edu

## I. Abstract

We address the limited receptive field and intolerance to cold-start in state-of-the-art matrix completion methods by providing a generalized framework, a novel graph fusion layer, and an analysis of the cold-start problem. We focus on the recommendation systems matrix completion problem. Specifically, our goal is to predict what ratings a user will give an item, based on past user rating history. Our model outperforms the state-of-the-art method and shows the importance of separating different entity types prior to performing graph convolution, as well as the importance of dropout.

## II. Introduction

Recommender systems have seen large improvements recently with the rise of social media and online shopping. Better user and product recommendation algorithms are sought after in order to provide users with preferred items.

Early methods for recommender systems fall under two categories: 1) content based filtering and 2) collaborative filtering. In content based filtering, user and item features are used to predict a user's preferences. In collaborative filtering, a user's preferences are predicted by using the preferences of similar users [1]. The collaborative filtering problem can further be simplified to **matrix completion**: where the known ratings are used to predict the unknown ratings off an user-item rating matrix. Matrix completion is a well studied problem with many principled methods such as **S**ingular **V**alue **D**ecomposition (SVD) which performs matrix factorization to find a user matrix $w$ and a height matrix $h$. The product of these two matrices is the predicted rating matrix. However, these models lack representational power and do not work adequately on large real world datasets.

Recent methods for recommendation systems adopt **G**raph **N**eural **N**etworks (GNNs), which are node feature learning algorithms that incorporates structural information into rich node embeddings. Particularly, GNNs are used to encode relationships between users and items into user or item embeddings. GNN-based recommendation systems perform recommendations either by solving the link prediction problem or the two-tower problem.

The link prediction problem treats the user-item rating matrix as an adjacency matrix of a user-item graph. This formulation achieves state-of-the-art performance; however, current solutions use outdated forms of message passing, which does not account for user-user or item-item interactions. Furthermore, they fail to address the cold start issue, where the user-item interaction graph may have empty rows or columns.

The two-tower model uses a GNN on the user-user and item-item graphs to learn user and item embeddings. The rating assigned to each user and item pair is then the inner product of the specified user and item embeddings, which is later used for recommendation; however, current solutions to the two-tower problem neglect existing user-item relationships, which has been shown to improve performance.

Building upon these ideas, we propose **G**raph **AT**tention **M**atrix **C**ompletion (GATMC) which provides a general framework for matrix completion. In our formulation, we also propose a novel **G**raph **F**usion **N**etwork (GFN) which fuses multiple user-user, item-item, and user-item graphs to fully incorporate both collborative and content based informations into rich embeddings. To handle cold start, we apply dropout on the edges of the user and item nodes during training. We present four main contributions:

- An reimplementation of **G**raph **C**onvolutional **M**atrix **C**ompletion (GCMC), the state-of-the-art link prediction recommendation system.
- A generalized framework, GATMC, which uses a novel GFN layer to perform link prediction.
- Training user-user, item-item, and user-item graphs instead of only a user-item bipartite graph.
- Analyzing the effects of dropout on cold start users.

## III. Related Works

Several works are used as a reference for using GNNs to implement recommendation systems and addressing the cold start problem.

DropoutNet, proposed by Volkovs et al., generalizes a deep neural network for cold start while still maintaining warm start accuracy [2]. This is done by utilizing dropout, which involves selecting random users and items per batch and setting their preferences to 0.

A recent survey on graph neural network techniques on recommendation systems [3] show that the user-item relationships lends itself naturally to a bipartite graph. These user-item bipartite graphs are often enhanced using user social graphs, and knowledge graphs.

One approach to addressing the cold start problem is to use hierarchical graph embeddings. Proposed by Maksimov et al., [4] HGE uses a hierarchical graph to obtain stable item-item embeddings for each level. Each hierarchy level

corresponds to an item-item graph, and information flow between layers is carried out through the skip-connection mechanism.

Another approach to addressing the cold start problem is by using attribute graph neural networks. AGNN, proposed by Qian et al., constructs an attribute graph instead of a traditional user-item bipartite graph [5]. This addresses the cold start problem in that the missing preference embeddings can be reconstructed from the attribute distribution of users by using a variational auto-encoder.

KGCN, or Knowledge Graph Covolutional Network, proposed by Wang et al., uses item-item knowledge graphs to enhance a user-item graph [6]. The attribute information of an item-item knowledge graph is mined to obtain latent connection to other related items.

Graph Convolution Matrix Completion, proposed by Berg et al., frames matrix completion for recommender systems as a link prediction problem for graph networks [7]. A graph auto-encoder constructs latent user-item features through message passing on the user-item bipartite graph. These latent features are then used to construct rating links through a bilinear decoder. We aim to improve upon this model and extend this to the cold-start setting.

## IV. Problem Definition

As adopted in various works, we treat recommendation systems as link prediction on bipartite user-item graphs. Specifically, we define a user-item graph as $\mathcal{G} = (\mathcal{W}, \mathcal{E}, \mathcal{R})$, where $\mathcal{W} = \mathcal{U} \cup \mathcal{V}$ is the set of all nodes, composed of a union of a set of users, $\mathcal{U} = (u_1, u_2, ..., u_M)$, and a set of items, $\mathcal{V} = (v_1, v_2, ..., v_N)$. $\mathcal{E}$ is the set of edges, where each edge, $(u_i, v_j, r) \in \mathcal{E} : u \in \mathcal{U}, v \in V, r \in \mathcal{R}$, is a tuple of a user node, item node, and rating. $\mathcal{R} = (1, 2, ..., R)$ is the set of possible ratings that a user can give an item.

Our recommendation problem is to find the rating, $r$, that a user $u_i$ would give an item $v_j$, which is equivalent to predicting the existence of $(u_i, v_j, r)$ links that do not already exist in the dataset. Formally, we are provided with an incomplete graph, $\mathcal{G}' = (\mathcal{W}, \mathcal{E}', \mathcal{R})$, and our goal is to find the full graph, $\mathcal{G} = (\mathcal{W}, \mathcal{E}, \mathcal{R})$ where the nodes and ratings between the two graphs are the same, but the incomplete graph only has a subset of all the edges of the true bipartite graph, whose edges include all combinations of users and items, $\mathcal{E}' \subset \mathcal{E} = (u_i, v_j, r) \forall u_i, v_j \in \mathcal{U} \times \mathcal{V}$.

We discover the whole graph $\mathcal{G}$ by discovering the missing edges, $\mathcal{E} \setminus \mathcal{E}'$, and more specifically, the ratings of user-item pairs that do not exist in the incomplete graph. We propose that this rating can be formulated as a multinomial distribution over the items given a user, and we wish to learn a model, $f(\cdot)$, which can determine which rating which is most likely for a given user-item pair, $r = f(u_i, v_j | \mathcal{G}')$. Notice, this formulation can be applied to both known edges in $\mathcal{G}'$ as well as unknown edges in $\mathcal{G}$. Hence, we present an autoencoder model trained on $\mathcal{G}'$, which can discover missing links in $\mathcal{G}$. To ensure the representational power, we apply the latest techniques in graph neural networks and attention mechanisms.

## V. Graph Attention Matrix Completion

In this section, we will propose GATMC by first covering the overall model architecture, then specific components, and finally additional design features. The overall data flow of the model is described in Figure 1.

### A. Overall Model Architecture

The goal of GATMC is to learn a function that maps node pairs to ratings, $f : \mathcal{U} \times \mathcal{V} \to \mathcal{R}$; however, because the ratings, $\mathcal{R}$, is a discrete finite set, we can transform this problem into finding a function that computes the multinomial probability of a certain rating given a user-item pair, $g : \mathcal{U} \times \mathcal{V} \times \mathcal{R} \to [0, 1]$. To satisfy the rules of probability, we normalize the outputs of some other network, $h : \mathcal{U} \times \mathcal{V} \times \mathcal{R} \to \mathbb{R}$, which can return any real number with a softmax function over the possible ratings, $g(\cdot) = Softmax_{\mathcal{R}}(h(\cdot))$.

Recently, there has been much success in constructing very rich features and representations of entities through the application of deep learning [8]. Specifically, graph neural networks has been widely applied to obtain powerful representations of nodes through a relaxed version of spectral graph convolutions [9]. These methods output a feature for each individual node, or in the case of recommendation systems, users and items. To be able to harness these rich features we adopt a two-stage learning pipeline. In the first stage, GATMC will learn **feature representations**, a $D$-dimensional vector, for every user and item with a neural network, $\phi_\theta$. In the second stage, GATMC will compute a **kernel**, $K_r$, on the rich feature representations to assign a unique score to every user-item pair. This is formally described in Equations (1) and (2).

$$Z = \phi_\theta(\mathcal{W}, \mathcal{E}', \mathcal{R}) : Z \in \mathbb{R}^{|\mathcal{W}| \times D} \tag{1}$$

$$h(u_i, v_i, r) = K_r(Z_{u_i}, Z_{v_i}) \tag{2}$$

Note, in general, $K_r$ can be any sort of kernel, and $\phi_\theta$ can be any sort of neural network. **G**raph **C**onvolutional **M**atrix **C**ompletion (GCMC) is a specific example of this paradigm, where the feature learning component consists of creating embeddings for every edge type and aggregating the edge embeddings that are linked to each node. The results are passed through a **M**ulti-**L**ayer **P**erceptron (MLP), and the kernel learning component consists of a different bilinear layer for each unique rating.

### B. Feature Learning

A major limitation of existing methods is that they either fail to fully leverage the graph's information or requires tinkering in order to do so. In the case of GCMC, label propagation is analogous to a single layer graph neural network, where each feature only absorbs information from its 1-hop neighbor. Because the user-item graph is bipartite, this essentially means the user embeddings only absorb information from items, and the item embeddings only absorb information from users. We hypothesize that user embeddings should also have a contextual awareness of other users that share a similar preference. A simple solution would be to stack multiple layers of GNNs;

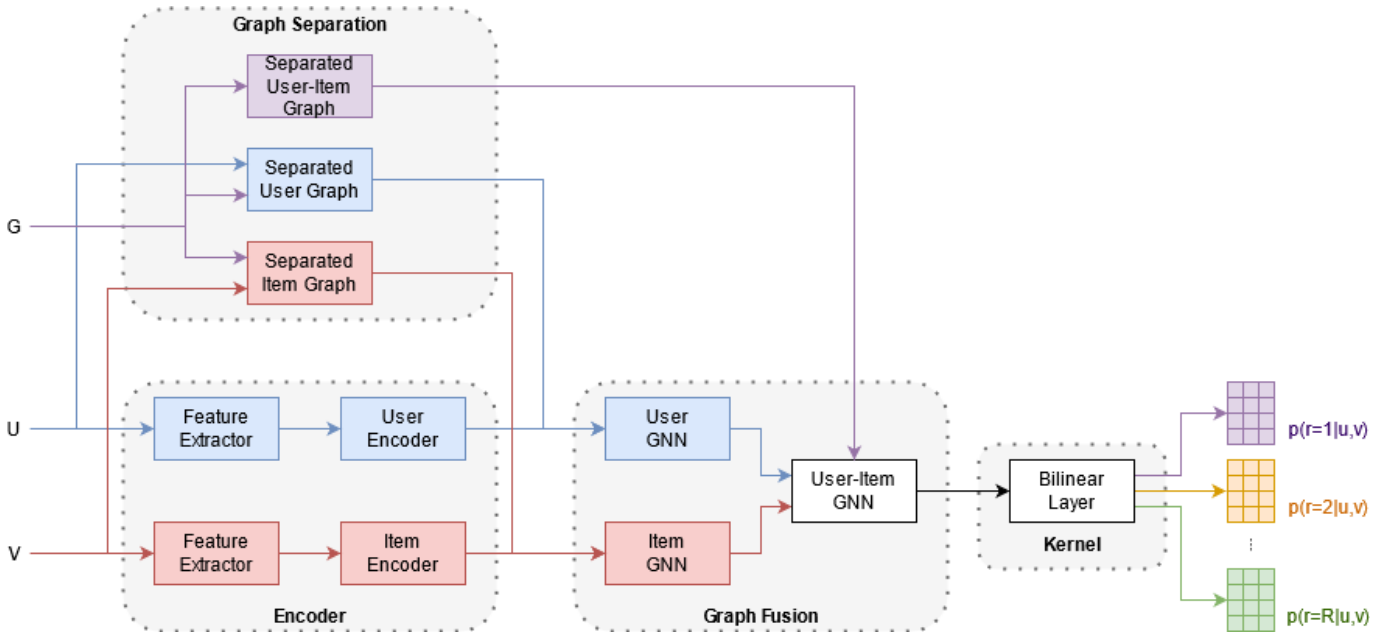



Fig. 1. Model Architecture: The graph separation encoder and graph fusion modules compose the feature learning portion. The kernel method compose the kernel learning portion. The input is the original graph $\mathcal{G}'$ and the output are probability maps for each item, which describes $h : \mathcal{U} \times \mathcal{V} \times \mathcal{R} \rightarrow \mathbb{R}$.

however, empirical results show that single layer graph neural networks outperform multi-layer networks for this task [7].

To overcome this challenge, our solution is to adopt single layer graph neural networks on multiple graphs. We suppose that current graph neural networks are not powerful enough to handle the intricate relationship of bipartite graphs [10]. Thus, instead of message passing between users and items with no distinction, we propose to run a separate graph neural network on user-user graphs, $\mathcal{G}_{ur}$ and item-item graphs, $\mathcal{G}_{vr}$, then finally merge the resulting embeddings of each with a final network on the user-item graph, $\mathcal{G}'$.

Specifically, we form the user-user graph, $\mathcal{G}_{ur}$, by linking all users who are 2-hop neighbors about the same relation, $r$, in the given user-item graph, $\mathcal{G}'$. We form the item-item graph, $\mathcal{G}_{vr}$, by running the same policy on items. Furthermore, we separate $\mathcal{G}'$ into $|\mathcal{R}|$ different graphs where each graph, $\mathcal{G}'_r$, is the same as $\mathcal{G}'$, except it only contains edges with relation $r$. We denote this step the **Graph Separation** step.

As the given graph, $\mathcal{G}'$ does not assume any extra information about the users and items apart from their connectivity, directly applying a graph neural network would ultimately amount to learning the degree distributions of different nodes. However, we notice that the local degree profile and connectivity statistics of a node can act as a unique feature embedding for that node. For instance, the average rating of nodes connected to a user indicates how strict/generous that user is. The number of ratings connected to a user indicates how prolific that user is. Note that these statistics have different meanings when applied on items. The average rating of an item indicates the quality of that item. The number of ratings connected to an item indicates how popular that item is. Thus, we collect the local degree statistics of each node (average rating, number of edges, bias scalar) to form the initial node features, $X \in \mathbb{R}^{M+N\times 3}$. We then apply a distinct MLP, depending on whether the node is a user or item, on the initial feature to form an encoding for each node $H = \sigma(X_{\mathcal{U}}W_{\mathcal{U}})||\sigma(X_{\mathcal{V}}W_{\mathcal{V}}) : H \in \mathbb{R}^{(M+N)\times d}$, where $X_{\mathcal{I}}$ denotes the elements in $X$ that corresponds the a set of nodes, $\mathcal{I}$, $W_{\mathcal{I}}$ denotes the weights of the MLP layer, and $\sigma(\cdot)$ is any activation function. In our case we set the $\sigma(\cdot)$ to be the identity function. We denote this step the **Encoder** Step.

After the **Graph Separation** and **Encoder** steps, we are presented with $|\mathcal{R}| \cdot (M \cdot N + 1)$ distinct graphs and initial encoding for users and items, $H \in \mathbb{R}^{M+N\times d}$. To fuse the latent structure of each separated graph, we run multiple **G**raph **F**usion **N**etwork (GFN) stacks. Formally, a GFN first propagates the graph encodings on the user-user and item-item graphs in parallel, and then utilizes the bipartite graph and runs a final round of propagation on the user-item graph. This process is repeated for each relation type, $r$, and then the final node embeddings for each relation type are fused together through an aggregation function. This can be anything from attention to simple summation. This process is formally described in Equations (3) to (6), denoted as the **Graph Fusion** step.

$$Y_{\mathcal{U}r} = GNN_{\mathcal{U}r}(H_{\mathcal{U}}) \tag{3}$$

$$Y_{\mathcal{V}r} = GNN_{\mathcal{V}r}(H_{\mathcal{V}}) \tag{4}$$

$$Y_r = GNN_{\mathcal{W}r}(Y_{\mathcal{U}r}||Y_{\mathcal{V}r}) \tag{5}$$

$$Z = AGG(||_r Y_r) \tag{6}$$

In our experiments, we set the aggregation function to be a concatenation of the final embeddings followed by a MLP

layer. We experimented with both Graph Convolution Network (GCN) and Graph Attention Network (GAT) as the GNN propagation architecture on the separated graphs.

### C. Kernel Learning

As used in many popular works [7], we use a bilinear layer as our kernel function, as denoted in Equation (7) where $Z_{u_i} \in \mathbb{R}^d$ is the embedding for node $u_i$ and $Z_{v_j} \in \mathbb{R}^d$ is the embedding for node $v_j$. Notice, the resulting output is $|\mathcal{R}|$ probability maps, each of which are a $|\mathcal{U}| \times |\mathcal{V}|$ matrix of real numbers. These probability maps act as our probability mapping function, $h : \mathcal{U} \times V \times \mathcal{R} \to \mathbb{R}$. Other kernels, such as Neural Tangent Kernel, Polynomial Kernel, or Gaussian Kernel, may also be explored, but we leave this as future work.

$$K_r(Z_{u_i}, Z_{v_j}) = (Z_{u_i})^T M_r(Z_{v_j}) \tag{7}$$

### D. Dropout, Regularization, and Residual Connections

To deal with the cold start problem, we also apply dropout on the edges of the initial bipartite graph. This simulates the case where additional initial edges are missing from the graph and helps with overfitting and robustness.

We believe L2 regularization and residual connections between the graph embedding and encoding layers are the most effective methods to further improve overall model performance; however, due to the time restrictions, we leave it as future work.

### E. Loss Function

As we treat the ratings as a multinomial distribution, we adopt cross entropy loss to train our model to reproduce the links seen in $\mathcal{G}'$ given $\mathcal{G}'$, which is shown in Equation (8). We minimize this loss function through the ADAM optimizer. Notice, this makes GATMC a graph autoencoder framework which learns to generate edges in $\mathcal{G}'$, and hence also $\mathcal{G}$.

$$\mathcal{L}(\mathcal{G}') = \frac{-\sum_{(u_i, v_j, r) \in \mathcal{E}'} log(g(u_i, v_j, r))}{|\mathcal{E}'|} \tag{8}$$

## VI. Results

We conduct experiments on 5 different models: a PyTorch implementation of GCMC, GATMC with GCN layers, GATMC with GAT layers, and merging a user-item graph into the GFN layer for both of our models. Each model was run for 1000 epochs with a dropout of 0.7. The RMSE of each model averaged over 5 runs is reported below.

| Model | RMSE |
|---|---|
| GCMC | 0.979085 |
| GCN | 0.943471 |
| GCN + user-item graph | 0.940150 |
| GAT | 0.955823 |
| GAT + user-item graph | 0.950096 |

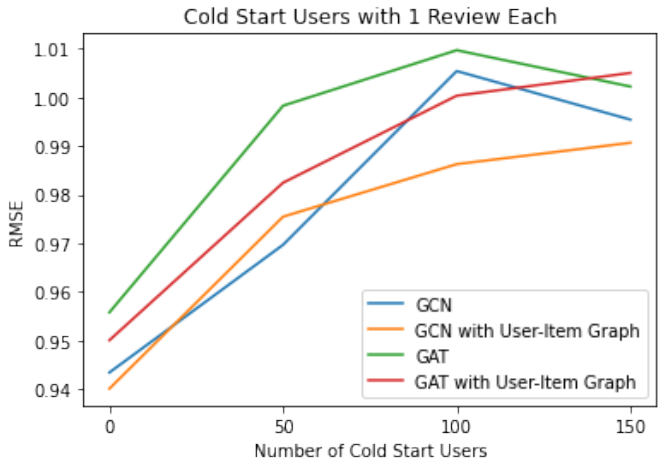
From the results, we see that all of our models outperformed the PyTorch reimplementation of GCMC by a significant margin. This is due to the use of user-user and item-item graphs, which is able to learn the user and item embeddings better over only using a user-item graph. We also see that using GCN layers in our model outperformed the GAT layers. This might be due to the GAT layer overfitting on the training data. By applying L2 regularization and residual connections, we believe that using GAT layers will produce a better result than using GCN layers. Lastly, we see that the use of a user-item graph in the GFN layer was able to improve the RMSE for both the GCN and GAT models. This enriches the user and item embeddings with more information, which results in more accurate user and item embeddings. Next, we explored the effects of varying the dropout for each of the models.

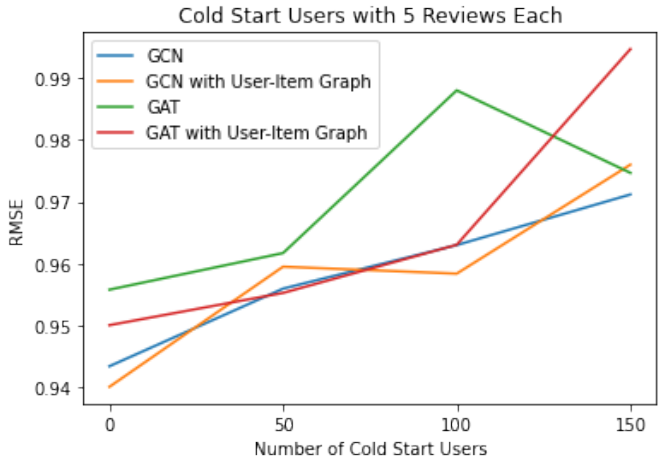
| Model | Dropout | RMSE |
|---|---|---|
| GCMC | 0 | 1.053848 |
| GCMC | 0.3 | 0.992191 |
| GCMC | 0.5 | 0.969046 |
| GCMC | 0.7 | 0.979085 |
| GCMC | 0.9 | 1.106354 |
| GCN | 0 | 0.944489 |
| GCN | 0.3 | 0.947893 |
| GCN | 0.5 | 0.947271 |
| GCN | 0.7 | 0.943471 |
| GCN | 0.9 | 0.956911 |
| GCN + user-item graph | 0 | 0.946388 |
| GCN + user-item graph | 0.3 | 0.946072 |
| GCN + user-item graph | 0.5 | 0.948600 |
| GCN + user-item graph | 0.7 | 0.940150 |
| GCN + user-item graph | 0.9 | 0.953188 |
| GAT | 0 | 0.958864 |
| GAT | 0.3 | 0.953672 |
| GAT | 0.5 | 0.951407 |
| GAT | 0.7 | 0.955823 |
| GAT | 0.9 | 0.959142 |
| GAT + user-item graph | 0 | 0.957445 |
| GAT + user-item graph | 0.3 | 0.949380 |
| GAT + user-item graph | 0.5 | 0.943916 |
| GAT + user-item graph | 0.7 | 0.950096 |
| GAT + user-item graph | 0.9 | 0.956317 |

From the results, we see that a dropout between 0.3 to 0.7 achieved the lowest RMSE. By varying the dropout parameter, we can fine tune each model to achieve a better RMSE. The GAT models were the most responsive to dropout, showing that they can be improved through other means of regularization as well, such as adding residual connections.

### A. Cold Start

We then explore the effectiveness of our models on the cold start setting. To simulate the cold start setting, we choose a random $N_c = \{0, 50, 100, 150\}$ number of users and removed all but $N_r = \{1, 5\}$ ratings from each user. The results are reported below:




Generally speaking, the RMSE increases as the number of cold start users increase. We also see that the cold start case with 1 review has higher RMSE than the case with 5 reviews. These results show that both the number of cold start users and the number of reviews for each cold start user has an impact on the accuracy of all models. We also see that as we increase the number of cold start users, the models that incorporate the user-item graphs were able to maintain better performance over their base counterparts.

### *B. Dropout vs Cold Start*

To analyze the effect of dropout on cold start, we ran each model with different dropout values on different amount of cold start users and reviews. The results are included in the appendix. Similarly to before, we found that the optimal dropout value ranged between 0.3 and 0.5. As mentioned previously, dropout was able to help in the cold start case by simulating missing reviews. By doing this during training, the model becomes regularized and robust, allowing it to perform better in cold start scenarios. However, we found that applying a dropout of 0.9 also increased the RMSE at many times. This is likely due to the model becoming too regularized and not learning enough from the training data.

## VII. Conclusion

We introduce GATMC, which is a general framework for link prediction recommendation systems. In our models, we train user-user and item-item graphs with a powerful graph fusion layer. With this architecture, GATMC was able to outperform a PyTorch implementation of GCMC. However, we also found that using GCN layers in our model instead of graph attention layers was able to produce even better results. We believe that this might be due the need for more regularization and also the choice in the aggregation function in the final embeddings. By using a different aggregation function such as stacking used in the original GCMC model, we believe that an even higher model accuracy can be achieved for GATMC. Lastly, we analyzed how cold start in recommender systems can be addressed by utilizing various dropout values.

## REFERENCES


[1] B. Schafer, B. J, D. Frankowski, Dan, Herlocker, Jon, Shilad, and S. Sen, "Collaborative filtering recommender systems," Jan. 2007.

[2] M. Volkovs, G. Yu, and T. Poutanen, "Dropoutnet: Addressing cold start in recommender systems," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017, pp. 4957–4966.

[3] S. Wu, W. Zhang, F. Sun, and B. Cui, *Graph neural networks in recommender systems: A survey*, 2020. arXiv: 2011.02260.

[4] I. Maksimov, R. Rivera-Castro, and E. Burnaev, *Addressing cold start in recommender systems with hierarchical graph neural networks*, 2020. arXiv: 2009.03455.

[5] T. Qian, Y. Liang, and Q. Li, *Solving cold start problem in recommendation with attribute graph neural networks*, 2020. arXiv: 1912.12398 [cs.IR].

[6] H. Wang, M. Zhao, X. Xie, W. Li, and M. Guo, "Knowledge graph convolutional networks for recommender systems," *The World Wide Web Conference on - WWW '19*, 2019. DOI: 10.1145/3308558.3313417. [Online]. Available: http://dx.doi.org/10.1145/3308558.3313417.

[7] R. van den Berg, T. N. Kipf, and M. Welling, *Graph convolutional matrix completion*, 2017. arXiv: 1706.02263 [stat.ML].

[8] L. C. Yan, B. Yoshua, and H. Geoffrey, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[10] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 3438–3445.

## VIII. Appendix