Graph Neural Networks for Collaborative Filtering

Raghu Radhakrishnan
Carnegie Mellon University
Pittsburgh, PA 15213
raghur@andrew.cmu.edu

Robert Pare
Carnegie Mellon University
Pittsburgh, PA 15213
rpare@andrew.cmu.edu

Michail Dontas
Carnegie Mellon University
Pittsburgh, PA 15213
mdontas@andrew.cmu.edu

1 Introduction

The high expressive power of graphical data structures makes them clear candidates for representing and analyzing data with complex relationships and dependencies. However, the use of graphs are not completely obvious and thoroughly explored when it comes to data inference and establishing relationships from a given dataset. We will focus on this aspect of graphical data structures throughout our investigation.

We concern ourselves specifically with the task of collaborative filtering. Collaborative filtering refers to a strategy in which we make predictions about the interests of a user by incorporating knowledge of other users and their known preferences. The underlying assumption for any collaborative filtering model is that if two users agree on a particular issue, it is likely that they will agree on other matters as well. Intuitively, this task of inferring relationships is one well-suited for a graphical structure.

Collaborative filtering is a task that plays a significant role in recommendation systems which are vital to the success of social media sites, content platforms, and e-commerce sites. Thus, the importance of such technologies is readily apparent and serves as the underlying motivation for our investigation. Ultimately, we aim to leverage the high-level representation power of graphs to establish these relationships and make inferences via Graph Neural Networks (GNNs) and compare these results with more standard neural network collaborative filters.

The dataset we will be using for our investigation is the Epinions dataset [9]. Epinions is a customer review website where users share reviews of products they have used. Each review consists of a user, an item, a rating from $1-5$ and a text review of the product. In addition to these reviews, the dataset also contains user-to-user trust relationships that when encoded in a graphical structure, we anticipate will augment our recommendations. For the purposes of our investigation, we exclude reviews written by users with fewer than 20 reviews are excluded. Table 1 below provides a brief summary of the dataset.

Table 1: Epinions User-Item

#Items	#Users	#Interactions	Sparsity
8999	335	10348	99.66%

For each of the models, we partition the dataset into a train-validation-test split of $80\% - 10\% - 10\%$. However, in order to ensure that any user in the validation or test sets appear in the training data, we take the last 20% of reviews for each user and split them evenly for the validation and test sets.

Using this dataset, we aim to tackle the collaborative filtering problem via a regression task. For each of our models, the inputs are a user and an item, and the output is a value which represents the predicted rating that the user would give for that particular item.

2 Background

2.1 Collaborative Filtering

Several techniques and architectures have been applied to the problem of collaborative filtering. Popularized by the Netflix Prize, one of the earliest standardized methods for collaborative filtering is known as matrix factorization (MF), which projects users and items into a shared latent space, using a vector of latent features to represent both users and items [2]. Thereafter, a user's interaction on an item is modelled as the inner product of user's latent vector and the respective item's latent vector. However, the inner product, which simply combines the multiplication of latent features linearly, may not be sufficient to capture the complex structure of user interaction data. In addition to MF, several traditional deep learning architectures have been used for modeling user-item interactions. In particular, MLPs and VAEs have been used as replacements to inner product methods [2], [4]. We investigate one such baseline method known as a Neural Collaborative Filter (NCF) [2].

2.2 Baseline Findings

Our baseline model, known as a Neural Collaborative Filter (NCF), consisted of an MLP based architecture to perform collaborative filtering. First, each user and item is represented as a one hot encoded vector. These vectors are then projected into a latent space via a linear transformation. During training and testing, the corresponding latent variables for a particular user and item are concatenated and passed into a simple two-layered neural network with ReLU nonlinearities. We then train the model to predict the rating for a user/item combination using mean-square-error loss. A pictorial representation of our architecture is shown in Figure 1.

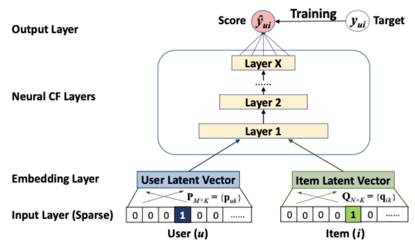
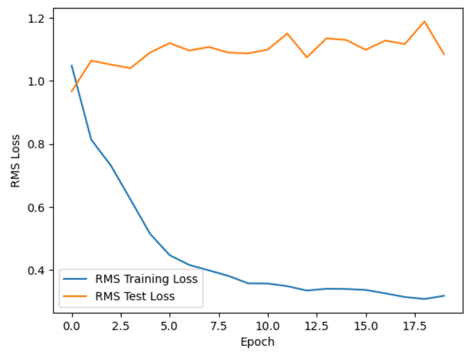
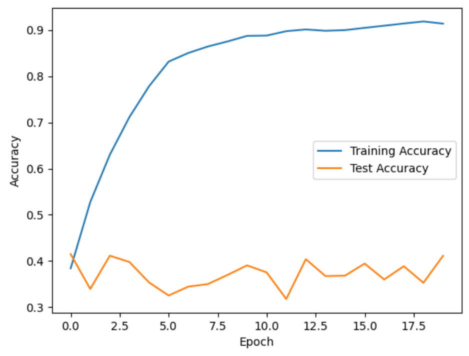


Figure 1: Neural Collaborative Filter Architecture [2]



(a) Training and test losses of NCF architecture.



(b) Training and test accuracies of NCF architecture.

Figure 2: Training and test curves achieved from the NCF architecture [2] on the Epinions dataset.

The original NCF paper [2] presented a strong argument for the Neural Collaborative Filter as a replacement for standard Matrix Factorization techniques. However, when applied to the Epinions dataset, we found that the architecture was unable to decode meaningful representations of the users and items and consequently could not achieve significantly better results on the test set than simply guessing a rating of 5 (the most common label) for each item. The results of our adoption of the NCF model are shown in Figure 2. Both our test accuracy and test loss seemed to stagnate throughout the entire training process implying that the representations being learned in training were not very representative of the task at hand.

From our results, it is apparent that our NCF implementation found great difficulty in expressing the users and items in the latent space. One potential reason could be limitations imposed by the dataset. After filtering for users with at least 20 reviews, the dataset size reduced to 10348 interactions. We propose that this quantity of data in addition to its sparsity made it difficult for the NCF architecture to extract meaningful representations in the latent space. However, we hypothesize that this is not an issue that the GNN will face to the same extent. This is due to the fact that the GNN will have better representation of the user-user interactions via direct user trust relationships as well as the two-hop neighborhood corresponding to other users that have reviewed similar products. Theoretically, this should yield superior results when encoding the users and items into a latent space.

3 Related Work

3.1 Neural Collaborative Filter [2]

One of the most promising deep learning approaches we encountered in the literature is the Neural Collaborative Filter method which we took inspiration from in our baseline NCF architecture. The paper provides a concise overview of the well-studied matrix factorization techniques but mainly focuses on experiments regarding various Neural based collaborative filtering models including the MLP model we utilized as our baseline approach. The paper found that their best NCF approach over various metrics was an ensemble method which combined the MLP method in our baseline with a generalized matrix factorization method and aggregated the results. However, the MLP approach did perform almost as well in all tasks with a much simpler architecture. The key insight from this paper is promise of deep learning approaches in the space of collaborative filtering. The nonlinearities induced in neural networks enables deep learning methods to learn more complex functions relating users and items and we aim to further these approaches via graphical structures. In our baseline approach, we adopted several ideas from this paper, namely the MLP filter, however we modified the task to a ratings prediction task rather than a ranking task.

3.2 Graph Neural Networks for Social Recommendation [7]

Much of the framework underlying our heterogeneous graphical architecture and model evaluations is derived from the Graph Neural Networks for Social Recommendation paper [7]. One of the main obstacles faced when designing a collaborative filtering system via a GNN is the heterogeneity of the task. Specifically, our graphical structure must be able to incorporate both user-item interactions as well as user-user interactions in order to fully exploit the representation power that the underlying graph presents. The problem of dealing with heterogeneous graphs in GNNs is not a completely solved problem, however the GraphRec framework presented in the paper proposes a method for dealing with this obstacle. Specifically, the paper divides the task of user and item modeling into two distinct tasks. In user modeling, they generate a separate embedding for the user with respect to their item interactions and with respect to their user interactions. These embeddings are concatenated to form the user's latent factor. In item modeling, for each item, its neighboring user embeddings are aggregated to form the item's latent factor. Finally, the two tasks are combined by concatenating the user latent factor and item latent factor from which we make predictions. This process is illustrated in Figure 3.

According to the paper, the GraphRec framework yielded promising results in comparison to matrix factorization and basic neural frameworks for collaborative filtering. It consistently achieved lower RMSE loss compared to several other baseline models. Given these results, we elected to adopt this general framework for our item recommendation system.

4 Methods

4.1 Graph Neural Network

The paradigm of interest underlying our investigation is the Graph Neural Network. A high level view of our graph based neural collaborative filtering model (GNCF) is shown in Figure 3. Our architecture consists of two edge encoders, one for modeling user-item interactions and another for modeling user-user social interactions, and an edge decoder which decodes a user-item interaction into a rating prediction. In the forward pass of our model, the user-item edge encoder learns a latent representation for users, which we call the item space user latent, and for items, which we call the item latent factor, via a round of message passing. Additionally, the user-user encoder learns another user representation, which we call the social space user latent, via a round of message passing. Then, the item space user latent and the social space user latent are concatenated together to form the final user latent factor. The user latent factor is then concatenated with the item latent factor, which is passed to the edge decoder. The edge decoder consists of a two layer MLP and a scaled sigmoid function so that the output of the edge decoder is a number between 1 and 5.

After testing several message passing routines combining different convolutional aggregators (GraphConv, GATConv, and SAGEConv) with varying neighborhood depth (1, 2, 3) we settled on the GraphSAGE method using a 2-hop neighborhood for aggregation[1]. GraphConv is a message passing algorithm inspired by the k dimensional Weisfeiler-Leman graph isomorphism heuristic [5], GATConv is message passing variant which applies self-attention to neighboring nodes during aggregation [6], and SAGEConv is a modified version of standard graph convolution message passing which is more suited to inductive learning [1]. The SAGEConv operator is formulated as

$$\mathbf{x}_i' = \mathbf{W}_1\mathbf{x}_i + \mathbf{W}_2 \cdot aggr_{j\in\mathcal{N}(\mathbf{x}_j)}\{\mathbf{x}_j\}$$

where $\mathbf{x}$ is the feature representation for a node, $\mathcal{N}(\mathbf{x})$ is the nodes in the neighborhood of $\mathbf{x}$, $\mathbf{W}_1$ and $\mathbf{W}_2$ are learnable weight matrices, and $aggr$ is the mean aggregator, the standard aggregator for SAGEConv.

4.2 Model Training

All parameters of the model were initialized using the default Pytorch geometric initializations. Moreover, the user embeddings were initialized as one-hot vectors and the item embeddings were initialized by encoding the name of the item with the all-MiniLM-L6-v2 sentence transformer from Hugging Face. Since we focused on ratings prediction, we optimized our model on mean square error loss, formulated as

$$Loss = \frac{1}{\mathcal{O}} \sum_{i,j\in\mathcal{O}} (\hat{r}_{i,j} - r_{i,j})^2$$

where $\mathcal{O}$ is the number of observed ratings and $\hat{r}_{i,j}$ and $r_{i,j}$ represent the predicted and true ratings respectively for user i on item j. To facilitate this optimization, we use the Adam optimizer and also include a batch normalization layer after the first linear layer in the edge decoder. Moreover, to help prevent overfitting, a perpetual problem in optimizing neural networks, we apply dropout with probability $p = 0.5$ after batch normalization in the edge decoder.

4.3 Hyperparameter Tuning

In order to optimize our model, we used the validation set to tune the learning rate, the number of hidden channels in the message passage convolution, the type of convolution operator for message passing, and the number of message passing layers. For the learning rate, we tested over the range $[0.01, 0.005, 0.001, 0.005]$. For the number of hidden channels, we tested over the range $[8, 16, 32, 64, 128]$. For the type of convolutional operator, we tested SAGEConv, GraphConv, and GATConv. For the number of message passing layers, we tested $[1, 2, 3]$. Unless otherwise noted, all plots are generated using a learning rate of 0.01, 8 hidden channels, the SAGEConv operator, and 2 message passing layers, which was the optimal hyperparameter configuration. Without special

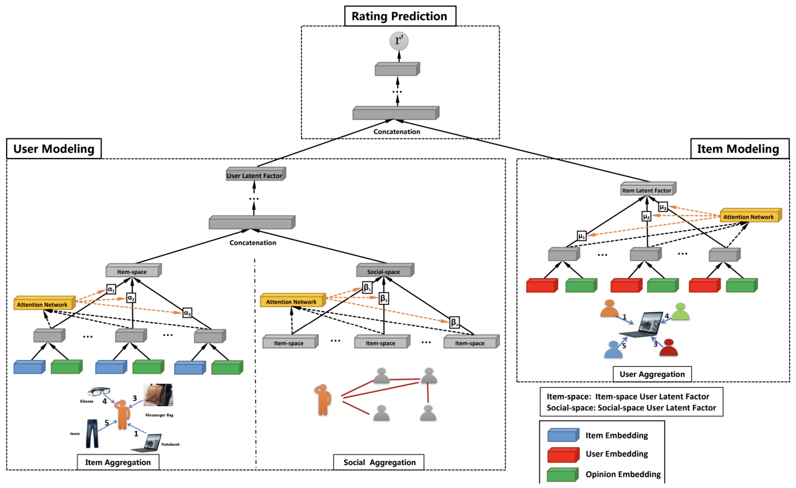


Figure 3: High-level architecture for GNN based collaborative filtering [7]

mention, the hidden layer of the edge decoder was given the same dimension as the number of hidden channels in the edge encoder and the same edge enccoder configuration was used for modeling both user-item interactions and modeling user-user social interactions.

5 Results

To evaluate and compare our models, we report root mean square error and accuracy. Since we formulate our optimization as a regression task, we use a standard rounding scheme to classify a prediction as “correct" or “incorrect". That is, if the predicted rating is within 0.5 of the ground truth, we consider the prediction correct and incorrect otherwise. Under the optimal configuration of hyperparameters (0.01 learning rate, 8 hidden channels, SAGEConv operator, 2 message passing layers), our model achieved a minimum root mean square error loss 0.9695 and a maximum test accuracy of 0.4624 while the baseline model achieved a minimum root mean square error loss of 0.9673 and a maximum test accuracy of 0.4148. Thus, our graph based model improved on the baseline accuracy by 11.47%, which is expected since graph neural networks excel at learning good feature representations and the GNCF also incorporates user-user social connections into its user latent representation. Additionally, our root mean square error loss using GNCF (0.9695) outperformed the reported root mean square loss of GraphRec (1.0631), a comparable method which also trained on the Epinions dataset [7]. Importantly though, we reduced the Epinions dataset to only include users with at least 20 reviews which reduces the complexity of the learning task, while GraphRec did not. A summary of the results across models is given in Table 2.

Table 2: Performance Comparison of Recommender Systems

	Models		
Metrics	NCF	GNCF	GraphRec
RMSE	0.9673	0.9695	1.0631
Accuracy	0.4148	0.4624	N/A

6 Discussion and Analysis

6.1 Analysis of Results

User Embedding Analysis The main advantage of graph based neural architectures is their ability to learn good feature representations. To gain some intuition on the quality of user representations

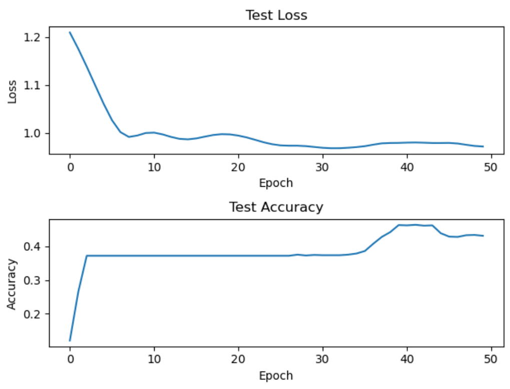
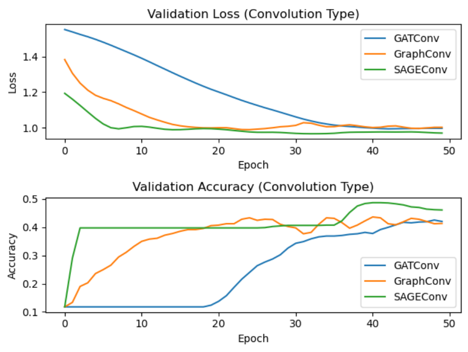
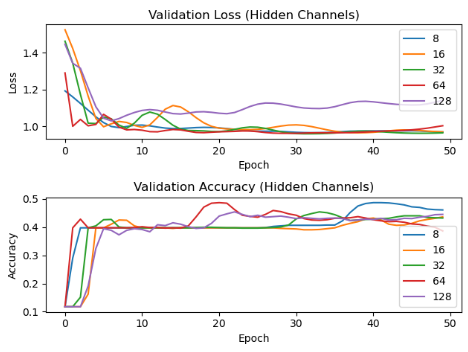


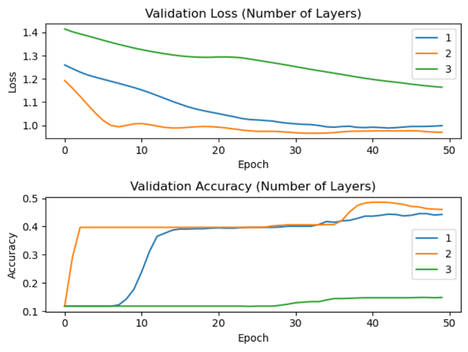
Figure 4: Test Loss and Accuracy for GNCF



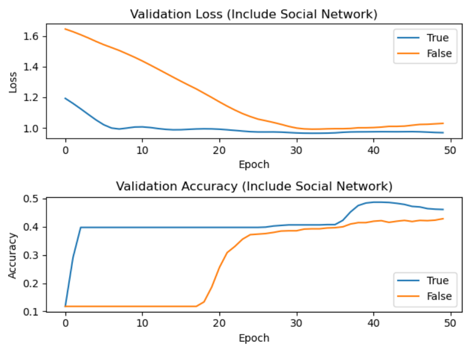
(a) Effect of Convolution Type on Loss and Accuracy



(b) Effect of Number of Hidden Channels on Loss and Accuracy



(c) Effect of Number of Layers on Loss and Accuracy



(d) Effect of Inclusion of Social Network on Loss and Accuracy

Figure 5: Effect of hyperparameters on Test Loss and Accuracy in GNCF

learned by the model, we located the users in our network which had the most similar and least similar feature representations to the first user in our dataset, who reviewed 45 items consisting mostly of cameras, printers, and monitors. We used the cosine similarity metric, which is formulated as

$$similarity(\mathbf{x}_i, \mathbf{x}_j) = \frac{\langle \mathbf{x}_i, \mathbf{x}_j \rangle}{||\mathbf{x}_i||_2 ||\mathbf{x}_j||_2}$$

where $\langle \cdot, \cdot \rangle$ is the dot product and $\mathbf{x}_i$ denotes the i^{th} user in our dataset. Since cosine similarity measures the cosine of the angle between two vectors in Euclidean space, a similarity of 1 corresponds to equivalent vectors, a similarity of 0 corresponds to orthogonal vectors, and a similarity of -1 corresponds to exactly opposite vectors. Among the users with the most similar embedding vector to $\mathbf{x}_0$ (i.e. similarity close to 1), we found that $similarity(\mathbf{x}_0, \mathbf{x}_{118}) = 0.9513$. Interestingly, user 43 reviewed 31 items consisting mostly of cameras and printers. Additionally, among the users with least similar embedding vector to $\mathbf{x}_0$ (i.e. similarity close to -1), we found that $similarity(\mathbf{x}_0, \mathbf{x}_{43}) = -.02567$, and user 43 reviewed 18 items consisting of Harry Potter and Star Wars themed items and Webkinz. By visual inspection, users 0 and 118 reviewed similar items and users 0 and 43 reviewed dissimilar items, which coincides with our intuition that user 0's learned feature representation is similar to user 118 and dissimilar to user 43.

Effect of Convolution Type Perhaps surprisingly since the SAGEConv architecture is the oldest and simplest among the three tested, the SAGEConv convolution offered the best performance among the three convolutional operators tested as shown in 5a. It is not entirely clear why SAGEConv offers the best performance and this question is left for future study.

Effect of Hidden Channels The number of hidden channels does not exhibit a clear trend on the performance of the model. As shown in 5b, 8 hidden channels provided the best performance. However, this performance benefit was only marginally better than other tested configurations. Moreover, there was no discernible trend between the number of the hidden channels and performance: for instance, 8 hidden channels had the highest test accuracy but 16 had the lowest, while 32, 64, and 128 hidden channels were somewhere in between.

Effect of Social Network As shown in 5d, including the social space user latent into the model confers a boost in performance. Since people acquire and disseminate information from their peers, we expect that users that are closely tied in the same social network will have similar taste profiles. Hence, we expect that combining this information with the user's item space latent will yield a better user representation.

Effect of Number of Layers As shown in 5c, the optimal performance was acheived when aggregating from a 2-hop neighborhood, while performance degraded somewhat for a 1-hop neighborhood and degraded significantly for a 3-hop neighborhood. Since a 2-hop neighborhood incorporates more neighbors into a given node's embedding, we expect a 2-hop neighborhood to yield a richer and more accurate feature representation than a 1-hop neighborhood. However, when using a 3-hop neighborhood, we incorporate *too* many neighbors into a given node's embedding and incur oversmoothing and oversquashing effects [3][8]. Oversmoothing occurs when unrelated nodes end up sharing much of the same neighborhood due to exponentially increasing neighborhood sizes, and oversquashing occurs when a large neighborhood has to be compressed into a fixed length node embedding size. As a result, distinct nodes have increasingly similar embeddings, which degrades the quality of the embeddings and model performance.

6.2 Model Limitations

Our model suffers from several key limitations including:

- **Collaborative Filtering Assumption:** Both of our models are constructed under the key collaborative filtering assumption. That is, we assume that user's with similar taste profiles (i.e. users that give similar reviews/ratings to similar items) should have similar feature representations. Hence, a good recommendation for one user is also a good recommendation to another user if the two users have similar representations. If this assumption is not satisfied, then our models will not produce any useful or meaningful results.
- **Limited data scope:** Deep learning approaches typically excel with large amounts of data. However, in this study we decided to trim the Epinions dataset substantially to only include users with at least 20 reviews since our baseline architecture, NCF, required users to have reviewed an ample number of items in order to learn somewhat meaningful user representations. In order to maintain some consistency between the baseline and the graph based approach, we decided to use the identically trimmed dataset for GNCF. We believe, however, that the performance of GNCF could be improved by training on the entire Epinions dataset or some other larger social recommendation based dataset.
- **Graph homophily assumption:** The social space user latent is formed under the assumption that the underlying social network is homophilous. That is, we assume that similar

individuals will aggregate in close neighborhoods and that the social connections formed always indicate some kind of “similarity" between users. However, online social networks are typically noisy. A user’s neighbors/friends in a social network may not indicate any kind of meaningful connection or could even indicate some kind of negative or dissimilar connection if some users are simply bots or fraudsters, which is often the case in real life online social networks.

- **Fixed network:** User-item rating predictions are learned and generated under the assumption that the network is fixed and unchanging. However, real life networks are often dynamically changing and social recommendation systems in industry must be able to adapt to new edges added or deleted from either the user-item review graph or the user-user social graph. Under our current framework, there is no convenient way to update user and item latent representations given alterations to the underlying graph structure without retraining the new network from scratch, which could be computationally prohibitive in a large scale setting.

6.3 Future Improvements

While our investigation yielded notable results regarding the power of graphical representations in deep learning, there is still much to gather in terms of further explorations in the space. These avenues of exploration include:

- **Dataset exploration:** As mentioned previously, we found it difficult to obtain meaningful results in our baseline implementation possibly due to a lack of information found in the dataset. We hypothesize that this issue also plagued our GNN implementation although we were still able to extract some meaningful information from the limited data. A future improvement to this investigation would be to expand to larger datasets with more connected graphical relationships. Intuitively, this would further accentuate the richer representations learned by the graphical network and strengthen the argument for GNNs in collaborative filtering systems.
- **Initialize richer edge embeddings:** One aspect of the dataset we elected not to use for complexity purposes was the actual text ratings provided by the users. Arguably this information could provide a richer understanding of a user’s interaction with a particular kind of item, and thus enhance our inference abilities. While we decided that a natural language understanding task may introduce too much variability and detract from the main purpose of our investigation, incorporating this extra information could lead to better insights into the power of GNNs.
- **Investigation of convolutional operators:** In our brief testing of various convolutional operators during message passing, we settled with the SageConv operator. Further testing may not only yield operators that are better suited for this domain but also reveal why a particular operator is better suited for a given task.

7 Teammates and Work Division

Robert Pare: Implemented baseline graph based architecture. Iterated on model development and incorporated social network embeddings into graph based architecture. Assisted with writing final report.

Raghu Radhakrishnan: Conducted preprocessing of Epinions dataset. Implemented and tested baseline architecture. Researched and proposed improvements to graph based architecture. Assisted with writing final report.

Michail Dontas: Performed hyperparameter tuning and training of models. Assisted with writing final report.

8 Code Access

All related code to run these experiments can be found on Github.

References

[1] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *CoRR*, vol. abs/1706.02216, 2017. arXiv: `1706.02216`. [Online]. Available: `http://arxiv.org/abs/1706.02216`.

[2] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, “Neural collaborative filtering,” *CoRR*, vol. abs/1708.05031, 2017. arXiv: `1708.05031`. [Online]. Available: `http://arxiv.org/abs/1708.05031`.

[3] Q. Li, Z. Han, and X. Wu, “Deeper insights into graph convolutional networks for semi-supervised learning,” *CoRR*, vol. abs/1801.07606, 2018. arXiv: `1801.07606`. [Online]. Available: `http://arxiv.org/abs/1801.07606`.

[4] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara, *Variational autoencoders for collaborative filtering*, 2018. arXiv: `1802.05814 [stat.ML]`.

[5] C. Morris, M. Ritzert, M. Fey, *et al.*, “Weisfeiler and leman go neural: Higher-order graph neural networks,” *CoRR*, vol. abs/1810.02244, 2018. arXiv: `1810.02244`. [Online]. Available: `http://arxiv.org/abs/1810.02244`.

[6] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph attention networks*, 2018. arXiv: `1710.10903 [stat.ML]`.

[7] W. Fan, Y. Ma, Q. Li, *et al.*, “Graph neural networks for social recommendation,” *CoRR*, vol. abs/1902.07243, 2019. arXiv: `1902.07243`. [Online]. Available: `http://arxiv.org/abs/1902.07243`.

[8] U. Alon and E. Yahav, “On the bottleneck of graph neural networks and its practical implications,” *CoRR*, vol. abs/2006.05205, 2020. arXiv: `2006.05205`. [Online]. Available: `https://arxiv.org/abs/2006.05205`.

[9] *Social recommendation data*. [Online]. Available: `https://cseweb.ucsd.edu/~jmcauley/datasets.html#social_data`.