Project Report: Graph Neural Network for Recommender System

Michael Chen
Georgia Tech
mchen389@gatech.edu

Ruiming Lu
Georgia Tech
rlu39@gatech.edu

Quang Vu
Georgia Tech
quangvu@gatech.edu

Abstract

Recommendation systems are everywhere in our daily life, from the next video we want to watch, to the next restaurant we will try out. Last few years have witnessed tremendous progress in the recommendation systems modeling, from simple logistic regression and gradient boosting algorithms, to the recent deep learning adoption. In this paper, we presented how graph neural networks can be applied to the recommendation system. We implemented and experimented two approaches (1) neural graph collaborative filtering (NGCF) (2) graph convolutional networks (GCN). The results look very promising. Graph neural networks leverage the topology information and can complement other algorithms in multiple components in a large recommendation system, for example, content retrieval and similar content recommendation.

1. Introduction, Background, & Motivation

1.1. What did you try to do? What problem did you try to solve?

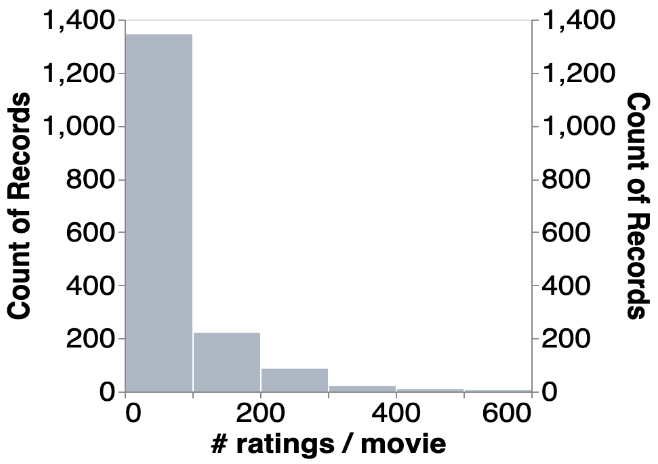
While deep learning has already made a huge impact on the recommendation system, we haven't seen wide adoption of graph neural networks yet. In this project, we tried to build a recommendation system using graph neural networks. Specifically, we try to predict what a user would rate a movie. While this is a direct use of GNN in the recommendation system, GNN can also be helpful in content retrieval and similar content recommendation.

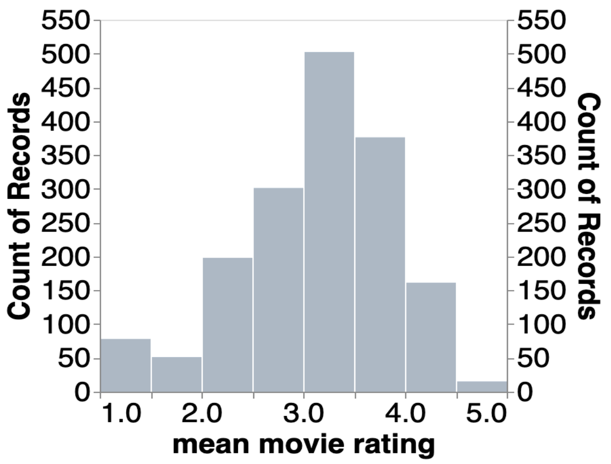
1.2. How is it done today and what are the limits of current practice?

The earliest approaches to recommender systems leveraged collaborative filtering. The idea behind that is to leverage how other users behave and find the most similar users to you. What they like then is likely what you like since your tastes are estimated to be similar to them. On the other hand, we can find the things similar to what you liked and recommend them to you. Those are the user-based and item-based collaborative filtering respectively. The drawback of this approach is that it only leverages the user interaction signals, while the users and content signals are completely missed.

Large scale linear regression and gradient boosting tree models are designed to make use of the users and content signals more efficiently. In those approaches, user features (e.g., demographics, view history) and content signals (e.g., categories, embeddings) are engineered before feeding into the model. The model is trained in a supervised fashion and the training targets are the implicit user feedbacks. As the feature space becomes increasingly large, deep neural networks now become the new standard as the model capacity is much larger than LR and GBDT. Additionally, DNN can extend naturally to a multi-head model if we want to model multiple user feedback at the same time (e.g., clicks, favourite, subscribe).

Alternatively we can perform content based recommendations. The caveat to that is that it requires extensive cataloging of the items and also doesn't recommend new trends that may occur.





1.3. Who cares? If you are successful, what difference will it make?

Netflix. Spotify. Any company or team that builds recommendation systems. There is already a flood of all kinds of media out there and it's only going to exponentially grow from here. Recommendation systems in theory will help cleanse this abundant data and provide good suggestions.

While there has been tremendous progressive in the last few years in this industry, mostly powered by the deep neural networks, graph neural networks only started seeing adoption until recently. GNN can be a great retrieval strategy that compliments current stream based and embedding based retrieval. Additionally, it is very helpful for similar content recommendations.

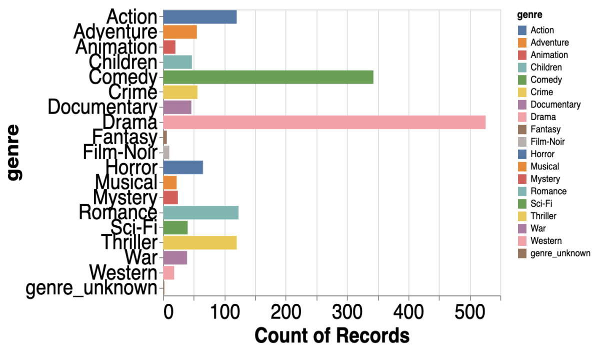
1.4. What data did you use?

We used the public MovieLens dataset [1].

(a) **For what purpose was the dataset created?**
Research and education.

(b) **Who created the dataset and on behalf of which entity?**
GroupLens Research created the dataset. GroupLens is a research lab and created the dataset on the behalf of the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities.

(c) **Who funded the creation of the dataset?**
The Department of Computer Science and Engineering at the University of Minnesota, Twin Cities.



2. Approach

What did you do exactly? How did you solve the problem? Why did you think it would be successful? Is anything new in your approach?

2.1 Neural Graph Collaborative Filtering

We wanted to build a recommendation system that given a user and a database of movies, we provide a movie that we think the user will like but haven't seen.

We used a generic collaborative filtering model as a baseline model to solve this problem. The underlying algorithm was matrix factorization. We compared this with a neural graph collaborative filtering (NGCF) [2] algorithm for our first comparison. We thought this would be successful because NGCF embeds the index of users and movies and also leverages neighboring nodes in the connected graph in order to augment and improve the model's representational power and accuracy. This new approach is leveraging the concept of neighborhood influence on each node to propagate embeddings.

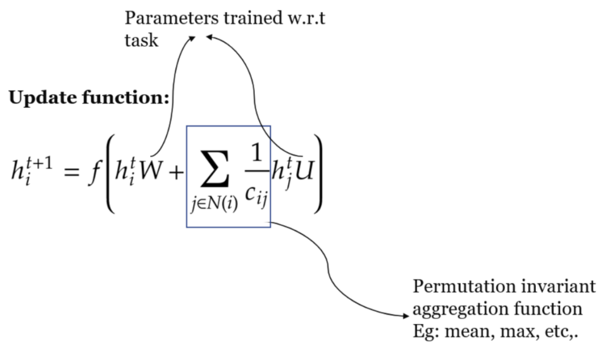
A bonus feature that we did not get to implement is the idea of a lightweight NGCF approach. Sometimes simplicity is best and the author of the LightGCN paper [3] states that by removing the non-linear activation function as well as the feature transformation matrices. What's left is simply a linear neighborhood aggregation of the embeddings and their paper goes to show improvement over traditional NGCF approaches on both the Gowalla and Amazon Book datasets.

2.2 Graph convolutional networks

We also used graph convolutional neural networks by themselves as another novel approach to this problem.

A graph represents a structure consisting of nodes representing entities and edges representing relationships between them. The topology encodes a lot of information that is missing in the traditional tabulated data format. Graph neural networks focus on learning algorithms that leverage this structure knowledge to get better performance on the downstream tasks. For our problem in particular, we model the users and movies as nodes, and ratings as edges, then the whole problem is formulated as an edge prediction problem.

The goal of the GNN learning algorithm is to learn better embeddings for the nodes and edges. Conceptually, GNNs iteratively aggregate feature information from neighbors to update the current representation of each node/edge. The aggregation function decides how the embedding from the neighbors are combined. We chose graph convolutional networks (GCN). The figure below illustrates how the representation of each node is updated in each interaction.


Parameters trained w.r.t task
Update function:
$h_i^{t+1} = f\left(h_i^t W + \sum_{j \in N(i)} \frac{1}{c_{ij}} h_j^t U\right)$
Permutation invariant aggregation function Eg: mean, max, etc,.

As the first step, we created a homogeneous graph, as in we don't differentiate the user node and movie node. We run the GCN on top of the graph, then predict the edge (i.e., ratings) by running a MLP on the concatenated embedding of the two nodes that the edge connects. Building on top of this setup, we add a new type of nodes, that is the genre nodes. A movie node and a genre node are connected if the movie belongs to the genre. This helps boost the performance a little more. For details about the results, please refer to the experiment and results section.

Ideally, we want to construct a heterogeneous graph where user nodes and movie nodes are treated differently. Additionally, we can use the move metadata (e.g., title, genre) to create the movie node attribution. For example, we can use pre-trained NLP models to create embedding based on the movie title and genre. Unfortunately, we don't get enough time to experiment with heterogeneous graphs thoroughly. We will leave it to future work.

2.3 Challenges

What problems did you anticipate? What problems did you encounter? Did the very first thing you tried work?

We expected problems in scaling out the algorithms to the bigger datasets that have 25M+ ratings, 50K+ movies, and 150K+ users.

We choose a homogeneous graph to solve this problem. We are concerned that homogenous graphs may not capture the rich difference between different types of data. However, a homogeneous graph is simpler and more intuitive as a starting point. Our approach works. But we believe heterogeneous graphs will allow us more control on propagating and updating, and more capable of incorporating new types of data.

3. Experiment & Results

Our experiment is conducted on the MovieLens small dataset, with 10,372 nodes and 100,836 ratings. The dataset is split into 70% train, 10% validation, and 20% test with our custom method such that only the `ratings` edges are split in each set, and all the nodes and other types of edge stay the dataset. This setup is to support our experiment in different ways to model our data.

Our first approach with a very simple GCN model (2 GCN & 1 dense layer) works. But as we work more on the problem and have better understanding about the data, we come up with more ideas to improve.

We implemented the algorithms in PyTorch frameworks. For GCN, we leverage PyTorch Geometric, which is a library built upon PyTorch to easily write and train GNNs for a wide range of applications.

3.1 Initial neural CF and NGCF approaches

We start off by attempting to replicate the traditional collaborative filtering approaches through neural embeddings. We call this approach the neural collaborative filtering model. The idea is that given a user and item pair, do we think the user will like the item? We embed the item index and the user index to their respective embeddings in order to represent the latent features. We then run the embedding outputs through some linear layers and output a score. This score represents how we think the user will like the item.

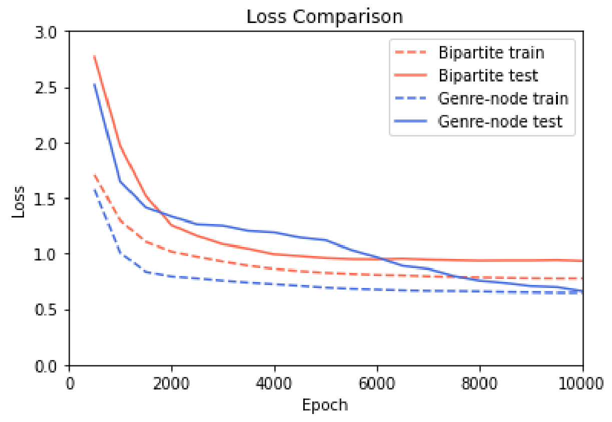
We take this a step further by applying GNN layers rather than just linear layers to create features from the initial user and item embeddings. We would still pipe these features into linear layers at the end and output a score that represents how the model thinks the user will respond to the item. This approach is our NGCF approach [2]. For these two CF approaches, we were leveraging just the connections between users and movies and did not use the other data such as genres.

Another notable thing to specify is that this approach is regression based as we are outputting a score given a connection between a user and a movie, rather than directly predicting an edge (given two nodes). CF approaches also tend to work best when the user has already been in the system and suffer from the cold start problem.

3.2 User-Movie vs User-Movie-Genre

Even though the objective is providing one of the 10 labels: 0.0-5.0, we treat this problem as regression. From our team's past experience, solving the regression problem, with the MSE as our loss function, will give a better loss gradient than using a classifier score.

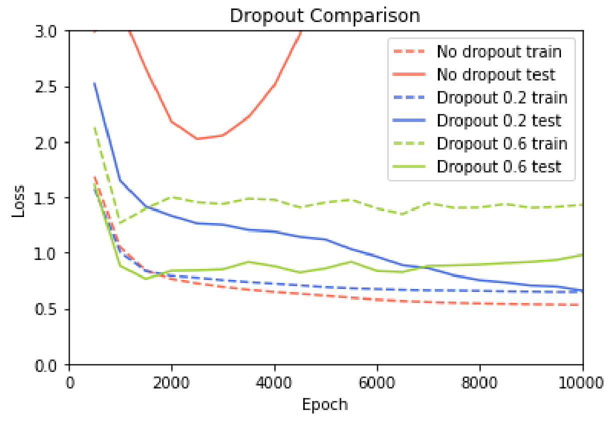
For our implementation, we use a homogeneous graph, treating all nodes as the same type with the same size of embedding. There are two modeling that we try in this project: treating the recommender system as bipartite graph of User->Movie, and treating Genres as separate entity: User->Movies->Genres.



In this experiment, the result matches our intuition on the problem. The bipartite graph is able to extract some relevant information from Movie's embed, and propagates that to Users. However, it falls short, comparing explicit modeling Genres as separate kinds of nodes. The bipartite graph only reflects the correlation between Users' rating and their Movie, so the Movie's embedding most likely represents the rating. By treating Genre as a separate kind of node, we bring in a new set of features to help connect Users' preference. This reflects on better predicting scores.

3.3 Dropout

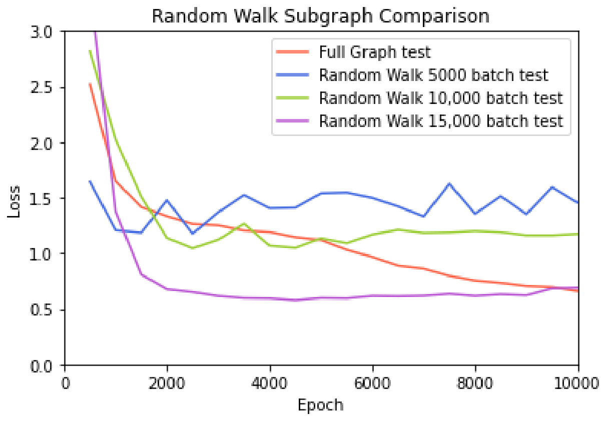
Dropout also shows to reduce overfitting on our model:



The dropout is applied between each GCN layer, and the linear regression layer. With no dropout, it clearly shows the overfitting, where test results get worse while training result is getting better. And with high dropout (ex: 0.6), the model loses too much information from the data.

3.4 Random Walk subgraph

From the beginning, we expect to have problems with large datasets. With over 220,000 nodes, and 25 millions ratings, it will push the limit of our GPU. Full graph training is only successful with our basic model. With a more tuned model, we need to sample the original graph into a batch of subgraphs with smaller size, in order to be able to avoid hardware limitation. While we successfully apply our approach on a large dataset, one training would take several hours to complete. Thus, we use results from small dataset to demonstrate the effect of random sampling subgraph.

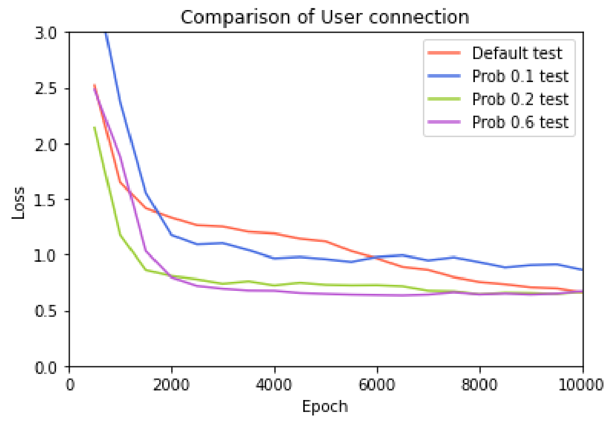


With larger batch size, we start to approach the optimal result of full graph training (10,372 nodes). With 5000 sampling batches, there are an average of ~4,800 nodes included. With 15,000 sampling batches, there are an average of ~8000 nodes included. Interestingly in our result, sampling 8000 nodes would result in faster convergence than full graph training, with the same optimal result. This matches our intuition about batch training: the larger the batch, the update will be closer to the mean of all losses, making it smoother but also slower to update the entire neural network.

We want to note that at some point the random walk sampling will bottle-neck on the CPU rather than GPU. Each sampling will involve computation from the CPU to figure out which nodes and edges are involved. The larger the batch, the longer the GPU will need to wait for the CPU to prepare the input. If time for preparing 1 input is longer than time to train 1 epoch, then increasing batch size will hurt training time.

3.5 Extra User Connections

In this experiment, we want to measure the effect if there is message passing between User nodes. Those "virtual" connections are not explicitly present in the data. The motivation for this experiment came from having residual connection in Deep network [5], but we want to apply it on our shallow Graph network. This is implemented by randomly adding a connection between two different User nodes, with a probability P.



The result shows that with extra connections between users, the training converges faster than our default baseline. In our result, it shows that above P=0.2, it does not improve the convergence of our training.

4. Work Division

Student Name	Contributed Aspects	Details
Michael Chen	Data creation and Implementation	- NGCF model implementation & analysis. - First to suggest using regression instead of classifier. - Discuss difference between link scoring vs link prediction.
Quang Vu	Data creation and Implementation Experimentation and Analysis	- Implement and tune GCN. - Implement custom train/test split, random walk subgraph, User connection.
Ruiming Lu	Data creation and Implementation	- Implement the training pipeline, and 1st version of GCN. - Tune initial set of hyperparams. - Led the discussion of homogeneous vs heterogeneous.

References

[1] Harper, F. Maxwell, and Joseph A. Konstan. "The MovieLens Datasets: History and Context." *ACM Transactions on Interactive Intelligent Systems*, vol. 5, no. 4, Dec. 2015, p. 19:1-19:19. *January 2016*, https://doi.org/10.1145/2827872.
[2] Wang, Xiang, et al. "Neural Graph Collaborative Filtering." *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, July 2019, pp. 165–74. *arXiv.org*, https://doi.org/10.1145/3331184.3331267.
[3] He, Xiangnan, et al. "LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation." *ArXiv:2002.02126 [Cs]*, July 2020. *arXiv.org*, http://arxiv.org/abs/2002.02126
[4] Rajamanickam, Santhosh. "Graph Neural Network (GNN) Architectures for Recommendation Systems." *Medium*, 17 Sept. 2021, https://towardsdatascience.com/graph-neural-network-gnn-architectures-for-recommendation-systems-7b9dd0de0856.
[5] https://pytorch-geometric.readthedocs.io/en/latest/
[6] He, Kaiming, et al. "Deep Residual Learning for Image Recognition." *ArXiv:1512.03385 [Cs]*, 1, Dec. 2015. *arXiv.org*, http://arxiv.org/abs/1512.03385.