Movie Recommender System Final Report

Introduction

Recommender systems have been a hot topic of research because they have very ubiquitous applications nowaday. A good way to visualize the interactions in a recommender system is using a **bipartite graph** with the users and items (movies in this case) as nodes, and edges between them indicating user-item interactions. Those interactions could be a user positively rating a recipe, or buying a product, or watching a video. The graph will be bipartite because users can be interested in items, but items and users can't be interested in other items or users, respectively. This graph representation enables us to employ powerful models, such as deep Graph Convolutional Networks (GCN).

Data analysis

In this assignment, I will be using the MovieLens 100K dataset, which contains 100,000 ratings by 943 users of 1682 items (movies). To ensure the quality of the dataset, each user has rated at least 20 movies (every user node has at least twenty edges).

I did a little research and came to the conclusion that it is better to take only those posts with a rating higher than 3. Also, I notice that data has some type of normal distribution:

```
Rating Distribution

rating
3    27145
4    34174
5    21201
```

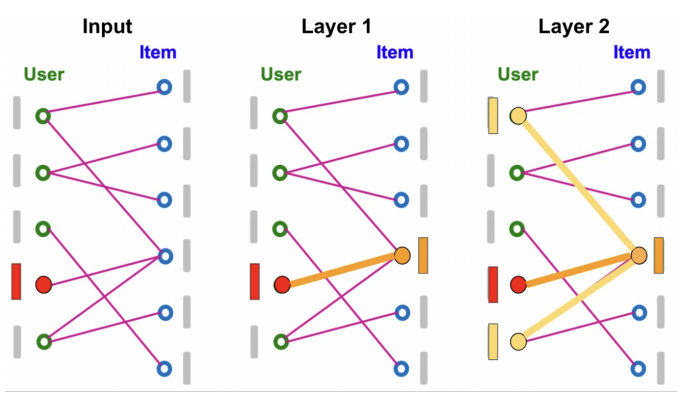
Model Implementation

Graph Neural Networks (GNNs) aim to learn structural and semantic patterns in input graph(s) relevant to various objective functions, similar to other deep learning domains. They work by passing messages among nodes along their edges and aggregating them for each node to update itself.

LightGCN

LightGCN is a neural graph model that focuses on the most important component of the network: neighborhood aggregation. Specifically, LightGCN only uses the initial node embeddings of the users and items in the graph as the learnable parameters of the network.

A simple way to visualize this is in the following figure, where users are on the left and items are on the right, and we see that the embedding of the highlighted node relies on nodes for K-hops away, depending on how many layers are used in the LightGCN model:



The new update rules for the user and item embeddings are now described by the following two functions:

$$\mathbf{e}_u^{(k+1)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} \mathbf{e}_i^{(k)},$$

$$\mathbf{e}_i^{(k+1)} = \sum_{u \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_u|}} \mathbf{e}_u^{(k)}.$$

$$\mathbf{e}_u^{(k+1)} = \underbrace{\sum_{i \in \mathcal{N}_u}}_{\text{Aggregate}} \overbrace{\frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} \mathbf{e}_i^{(k)}}^{\text{Message}}$$

For the message function, we note that each user node is simply gathering all of the information from the neighbor nodes, which are the item nodes.

Model Advantages and Disadvantages

Advantages of LightGCN

1. **Simplicity:** LightGCN offers a simpler architecture compared to traditional GCNs, making it easier to implement and train.

2. **Scalability:** The model focuses on the essential components of graph convolution and effectively leverages collaborative filtering without unnecessary complexity.

3. **Good Performance:** Despite its simplicity, LightGCN has demonstrated competitive performance in recommendation tasks.

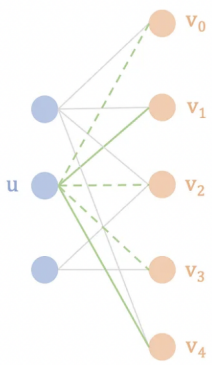
Disadvantages of LightGCN

1. **Limited Graph Information:** LightGCN only considers the user-item interaction graph, neglecting auxiliary information such as user and item features, which can be valuable in capturing richer contextual information.

2. **Lack of Embedding Differentiation:** Because LightGCN uses the same embeddings for both users and items, it may struggle to capture nuanced differences, leading to potential limitations in understanding fine-grained user and item preferences.

Training Process

Specifically, I use the Bayesian Personalized Ranking (BPR) loss as the surrogate. To understand BPR, we need to define the notion of positive and negative edges: positive edges are those that exist in the graph and negative edges are those that don't. In our bipartite graph, we can define for user ***u*** the set of all positive edges containing ***u***, ***E(u)*** and the set of all negative edges containing ***u***, ***E_neg(u)***.



positive edges for user u

$\{v_1, v\}$

negative edges for user u

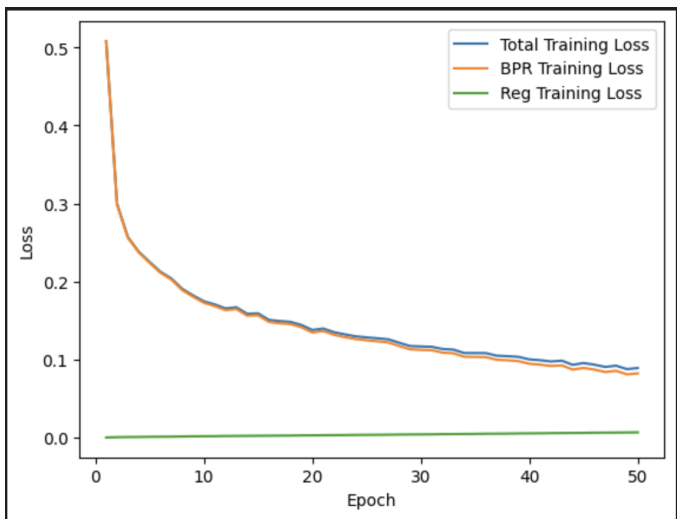
$\{v_0, v_2, v_3\}$

$$BPR(u) = \frac{1}{|E| \cdot |E_{neg}|} \sum_{(u,v_{pos}) \in E(u)} \sum_{(u,v_{neg}) \in E_{neg}(u)} -\log\left(\sigma(f_\theta(u, v_{pos}) - f_\theta(u, v_{neg}))\right)$$

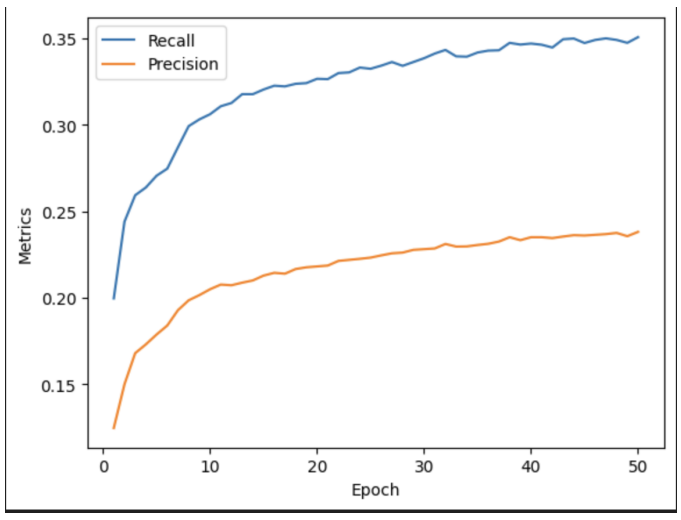
$$Loss = \frac{1}{|U|} \sum_{u \in U} BPR(u)$$

I split the dataset into an 80/20 train-test split and train both models using the minibatching procedure described above. The held-out edges in the test set are used to evaluate precision@K and recall@K after each training epoch:

Loss function - Bayesian Personalized Ranking(BPR)



Precision and Recall



Evaluation

For evaluation I used same metrics(precision and recall) for test dataset and got max results:

```
(base) kks02-osx2:movie_recomender_system kks02$ python benchmark/evaluate.py
Recall: 0.3508.
Preccision: 0.2381.
```

Results

In this assignment, I used graph neural networks to solve the problem of ranking/recommending films. I got the results of such metrics as Precission and Recall and got a very satisfying result, the model works well on the data provided in MovieLens 100K dataset.