From Matrix to Deep Knowledge Graph:

A multifaceted Approach to Recommendation Systems

team members: Tong Shenyang(leader), Wang Pengxin, Zhan Xiangcheng

> pre_video.mp4
>
> slides

From Matrix to Deep Knowledge Graph:
- A multifaceted Approach to Recommendation Systems
- Introduction
- Recommendation Algorithms
  - Typical recommendation algorithms:
    - Content-based filtering
    - Item-based collaborative filtering
    - User-based collaborative filtering
- Deep learning-based algorithms
  - Neural Collaborative
  - TransR Model
  - graph neural network-based recommendation algorithms
- Experiment
  - Neural Collaborative
    - Experiment Details
    - Results
    - Summary
  - TransR Model
    - Entity and Relation Embedding
    - Margin Loss and Negative Sampling
    - Experiment and Results
    - Training Details
    - Model Performance
    - Summary
  - graph neural network-based recommendation algorithms
    - Summary
- Conclusion
- Inference
- Appendix
  - Contribution

Introduction

A recommendation system is a powerful tool utilized to help users discover relevant items or content based on their preferences and behavior. It finds common usage across various online platforms like e-commerce websites, streaming services, and social media platforms. The main objective of a recommendation system is to furnish personalized and targeted suggestions to users, thereby enhancing their experience and aiding them in finding what they seek. These recommendations are formulated by analyzing user data encompassing previous actions, preferences, demographic information, and feedback.

Due to its significant impact in today's society, we have chosen a recommendation system as the focus of our final project. In this project, we will implement the following:

1. Typical recommendation algorithms: Content-based filtering, Item-based collaborative filtering, and User-based collaborative filtering
2. Deep learning-based algorithms: Neural Collaborative, graph neural network-based recommendation algorithms
3. Integration of knowledge graph into recommendation algorithms

We will explore and analyze the outcomes of each algorithm, comparing their effectiveness throughout the entirety of the project.

Recommendation Algorithms

Typical recommendation algorithms:

For both datasets, the approach to processing is quite similar. Hence, we will provide a detailed explanation of how we handled the `ml-1m` dataset for this particular algorithm.

Content-based filtering

1. First, import the data and use these three lists as headers for the data: `unames = ['user_id', 'gender', 'age', 'occupation', 'zip']`, `rnames = ['user_id', 'movie_id', 'rating', 'timestamp']`, `mnames = ['movie_id', 'title', 'genres']`.
2. Separate the genres of the movies into 18 different movie types. Consider these as (X, Y) pairs in a sparse matrix, where X represents the movie ID, and Y represents the movie genre. If a movie with ID X belongs to genre Y, then (X, Y) = 1.

```
movie_type_value.append((row['movie_id'],type_total.index(i),1))
```

```
['Animation', "Children's", 'Comedy', 'Adventure', 'Fantasy', 'Romance', 'Drama', 'Action', 'Crime', 'Thriller', 'Horror',
[(1, 0, 1), (1, 1, 1), (1, 2, 1), (2, 3, 1), (2, 1, 1), (2, 4, 1), (3, 2, 1), (3, 5, 1), (4, 2, 1), (4, 6, 1), (5, 2, 1),
```

1. Process the sparse matrix formed by the data points into a cosine similarity matrix to obtain the similarity matrix between movies based on their genres.

```
#Cosine similarity [-1,1] between (i,j)
cosine_sim = cosine_similarity(sparse_matrix)
print(cosine_sim.shape)
print(cosine_sim)
```

```
(3953, 3953)
[[0.         0.         0.         ... 0.         0.         0.        ]
 [0.         1.         0.33333333 ... 0.         0.         0.        ]
 [0.         0.33333333 1.         ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 1.         1.         0.70710678]
 [0.         0.         0.         ... 1.         1.         0.70710678]
 [0.         0.         0.         ... 0.70710678 0.70710678 1.        ]]
```

1. Choose any user, record the movies they already like as movie A, then sum all column values corresponding to the row of movie A in the similarity matrix. Sort these values to obtain the movies that the user might like. Here, the result is selecting user_id = 1, number = 5.

```
['Babe (1995)', 'Pollyanna (1960)', 'Wide Awake (1998)', 'Watership Down (1978)', 'Fluke (1995)']
```

Item-based collaborative filtering

1. First, import the data and use these three lists as headers for the data: `unames = ['user_id', 'gender', 'age', 'occupation', 'zip']`, `rnames = ['user_id', 'movie_id', 'rating', 'timestamp']`, `mnames = ['movie_id', 'title', 'genres']`.
2. Different from the previous algorithm, this time the sparse data point set (X, Y) comprises movie_id (X) and user_id (Y), where (X, Y) = rating, and rating is the user's rating for movie X.
3. Process this sparse matrix using cosine similarity to obtain the similarity matrix between movies based on ratings.
4. Select any user, record the movies they already like as movie A, then allocate weights based on the user's ratings for the movies. Sum up all the movie ratings.

```
for i in range(1,len(cosine_sim)):
    #print(i)
    _=0#sum
    __=0#devide
    for j in movie_indices:
        _+=round(cosine_sim[i][j]*weight_user_dict[j],5)
        __+=round(cosine_sim[i][j],5)
    if i in weight_user_dict:
        #print(i)
        user_weight_list.append(weight_user_dict[i])
    else:
        if __==0:
            user_weight_list.append(0)
        else:
            user_weight_list.append(_/__)
```

```
{1357: 5, 3068: 4, 1537: 4, 647: 3, 2194: 4, 648: 4, 2268: 5, 2628: 3, 1103: 3, 2916: 3, 3468: 5, 1210: 4, 1792: 3, 1
[-1, 3.7750062627410492, 3.710107165420833, 3.722930743914651, 3.7381744420575145, 3.725655588123848, 3.6944456545087
3953
The scores after sorting. [5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
The indices corresponding to the scores. [110, 318, 356, 380, 480, 515, 590, 593, 920, 1124, 1193, 1196, 1225, 1246,
["['Braveheart (1995)']", "['Shawshank Redemption, The (1994)']", "['Forrest Gump (1994)']", "['True Lies (1994)']",
```

User-based collaborative filtering

1. Same as above.
2. (X, Y) is opposite to Item-based collaborative filtering where X represents user_id and Y represents movie_id.
3. Process this sparse matrix using cosine similarity to obtain the similarity matrix between users based on ratings.
4. Sort by similarity matrix and identify users B, C, and D most similar to user A. Recommend movies liked by users B, C, and D to user A.

```
The indices of movies to recommend to 2: {1280, 1252, 3006}
["['Raise the Red Lantern (1991)']", "['Chinatown (1974)']", "['Insider, The (1999)']"]
```

Deep learning-based algorithms

Neural Collaborative

Neural Collaborative Filtering (NCF) is a recommendation algorithm that combines the power of neural networks with collaborative filtering techniques. It aims to provide personalized recommendations by modeling user-item interactions using deep learning architectures.

Traditional collaborative filtering methods rely on the latent factor representation of users and items to make recommendations. However, these methods may face challenges in capturing complex user-item interactions and handling sparse and noisy data.

We adopted three kinds of NCF algorithms.

- GMF (Generalized Matrix Factorization) is a variant of Neural Collaborative Filtering (NCF) that focuses on capturing the latent factors of users and items through a factorization process. It is a neural network model specifically designed for recommendation systems.
- MLP (Multi-Layer Perceptron) is a type of neural network architecture that consists of multiple layers of interconnected artificial neurons, also known as perceptrons. It is a versatile and widely used model that is capable of learning complex non-linear relationships in data.
- Neural Matrix Factorization (NeuMF) is a variant of Neural Collaborative Filtering (NCF) that combines the strengths of matrix factorization and multi-layer perceptron (MLP) models to improve recommendation accuracy. It was proposed as an enhancement to the GMF and MLP-based NCF models.

TransR Model

In the landscape of recommendation systems, accurately modeling the relationships between entities is crucial. The TransR model addresses this by embedding entities and relations into separate spaces, thereby enhancing predictive performance.

TransR distinguishes itself by considering entities and relations in separate spaces, allowing for more nuanced interaction modeling. Entities are first embedded in a common space and subsequently projected into relation-specific spaces, facilitating a more refined understanding of entity relationships.

graph neural network-based recommendation algorithms

For this algorithm, we adopted the LightGCN model proposed in 'LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation'. The paper analyzed NGCF and found that feature transformations and non-linear activations did not benefit collaborative filtering tasks. Surprisingly, removing these elements actually improved model performance. Consequently, the paper introduced a new structure, LightGCN, containing only the basic structure of GCN—neighborhood aggregation—for collaborative filtering.

- Regarding the handling of the ml-1m dataset, in summary, we grouped the user-movie-rating pairs in rating.dat by user and transformed them into the format user-[movie1, movie2,...].
- For processing the last-fm dataset, apart from similar processing to ml-1m, we needed to include additional data regarding relationships between users. This data was present in user_friends.dat and was imported into the model along with the rest of the data.

Experiment

Neural Collaborative

On one hand, the rating data was split into train and test datasets. On the other hand, we randomly pick 100 items which have no relationship to the users as test.negetive dataset.

Experiment Details

- Epochs: 20
- Learning rate: 0.001
- Batch size: 256
- Optimizer: Adam

Results

Model	ml-1m-HR	ml-1m-NDCG	Lastfm-HR	Lastfm-NDCG
GMF	0.7061	0.4078	0.9070	0.6598
MLP	0.7126	0.4267	0.8890	0.6554
NeuMF without pretraining	0.7217	0.4350	0.9033	0.6945
NeuMF with pretraining	0.7224	0.4303	0.9149	0.6607

Summary

NCF is a powerful technique that can greatly improve the performance of recommendation systems. By using neural networks to capture complex patterns and relationships, NCF is a more effective solution than traditional matrix factorization algorithms. The experimental results strongly support the adoption of NCF in recommendation systems. Its ability to capture complex patterns and relationships leads to significant improvements in HR and NDCG scores.

TransR Model

Entity and Relation Embedding

Entities are embedded into a high-dimensional space and then projected onto relation-specific spaces. This dual-embedding mechanism allows TransR to effectively capture the complexity of entity relationships.

Margin Loss and Negative Sampling

TransR employs negative sampling to introduce non-existent relations during training, making the model less prone to overfitting and more robust in predicting new relations. Margin loss is used to ensure that the positive examples are closer together compared to negative samples, within a defined margin.

Experiment and Results

The efficacy of TransR was evaluated on a subset of the LastFM dataset. The model was assessed using the Mean Reciprocal Rank (MRR), Mean Rank (MR), and Hit Ratio metrics.

Training Details

Training involved negative sampling and optimizing the margin loss to differentiate between positive and negative examples clearly.

Model Performance

The performance was quantified using several metrics:

Metric	MRR (raw)	MRR (filter)	MR (raw)	MR (filter)	Hit@10 (raw)	Hit@10 (filter)	Hit@3 (raw)	Hit@3 (filter)	Hit@1 (raw)	Hit@1 (filter)
l(raw)	0.037675	0.066404	1084.192	960.385	0.080818	0.063291	0.030521	0.026863	0.011088	0.025863
r(raw)	0.056162	0.097477	1213.0035	1191.152466	0.133516	0.198941	0.047516	0.100633	0.014480	0.043591
averaged(raw)	0.046919	0.081940	1148.5977	1075.768799	0.107167	0.168000	0.039018	0.081962	0.012784	0.035227

Summary

The TransR model demonstrates strong potential in recommendation system applications, effectively balancing precision and recall. The adoption of margin loss and negative sampling contributes to its ability to predict new user-artist interactions, providing a robust solution for dynamic and evolving recommendation environments.

graph neural network-based recommendation algorithms

Below are the results of our model on two datasets.

> all metrics is under
>
> topks=20
>
> decay=1e-4
>
> learning rate=0.001

> seed=2020
>
> recdim=32

- ml-1m stop at 50 epochs

	Recall	ndcg	precision
layer=1	0.0363	0.0375	0.02744
layer=2	0.0347	0.0377	0.02870
layer=3	0.0403	0.0481	0.03934
layer=4	0.0443	0.0569	0.04863

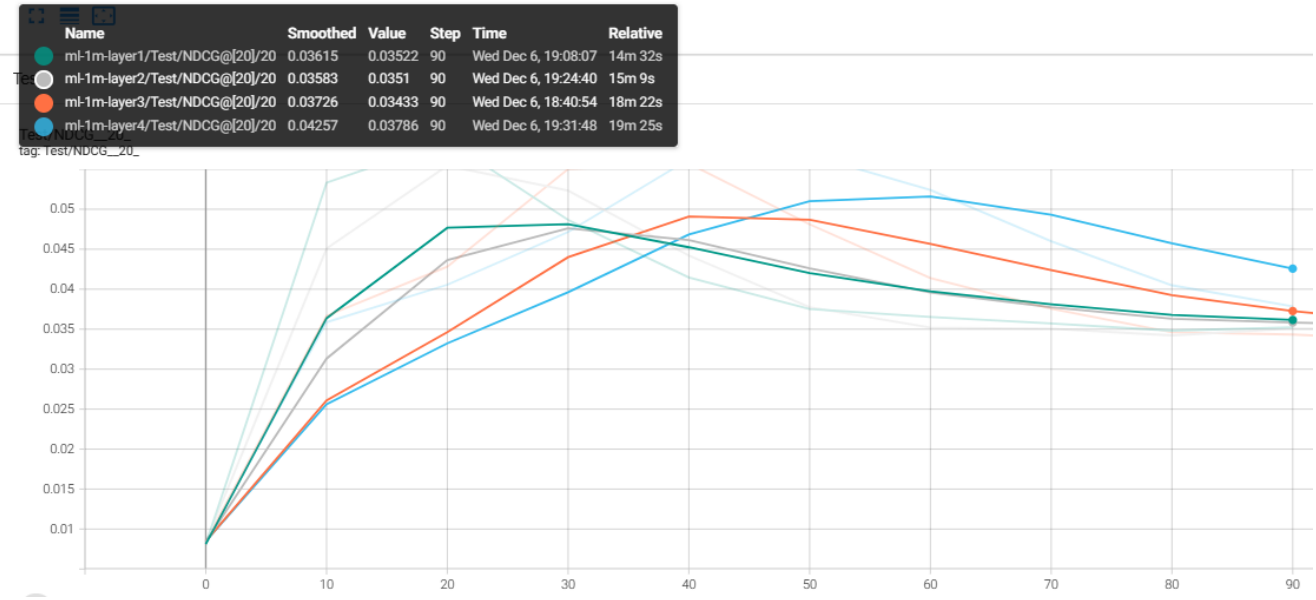
- ml-1m stop at 100 epochs

	Recall	ndcg	precision
layer=1	0.0356	0.0352	0.02457
layer=2	0.0348	0.0351	0.02481
layer=3	0.0331	0.0343	0.02548
layer=4	0.0343	0.0379	0.03433

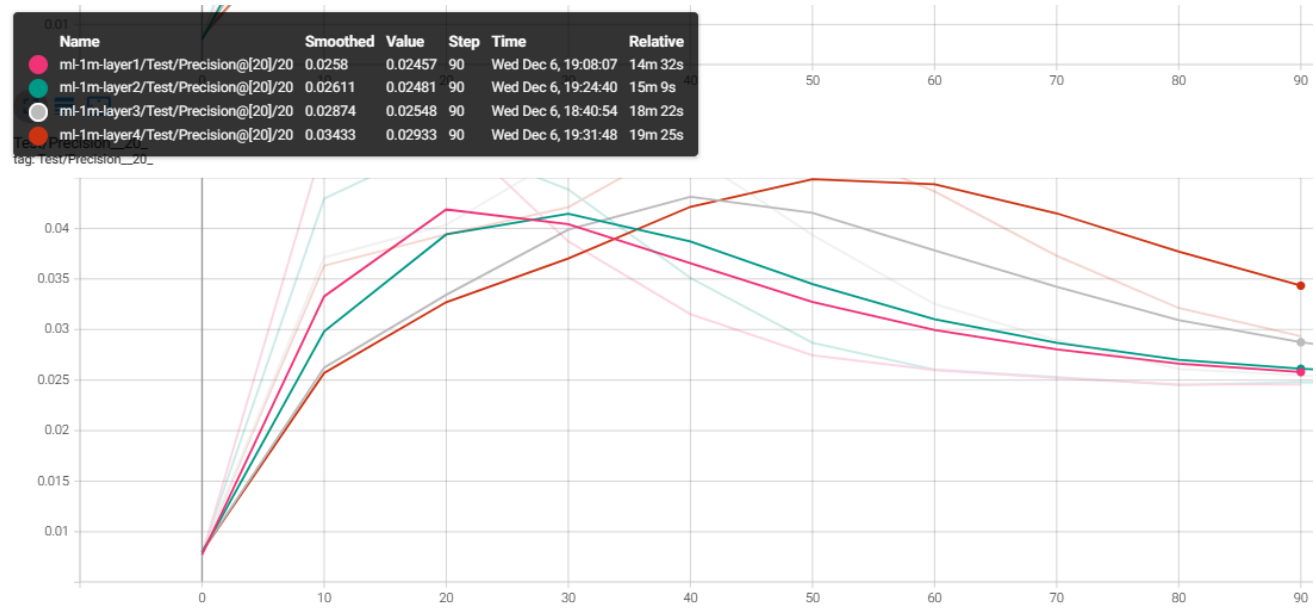
- hetrec2011-lastfm-2k stop at 100 epochs

	Recall	ndcg	precision
layer=1	0.2268	0.1690	0.06281
layer=2	0.2524	0.1926	0.07053
layer=3	0.2605	0.2027	0.07312
layer=4	0.2656	0.2051	0.07449

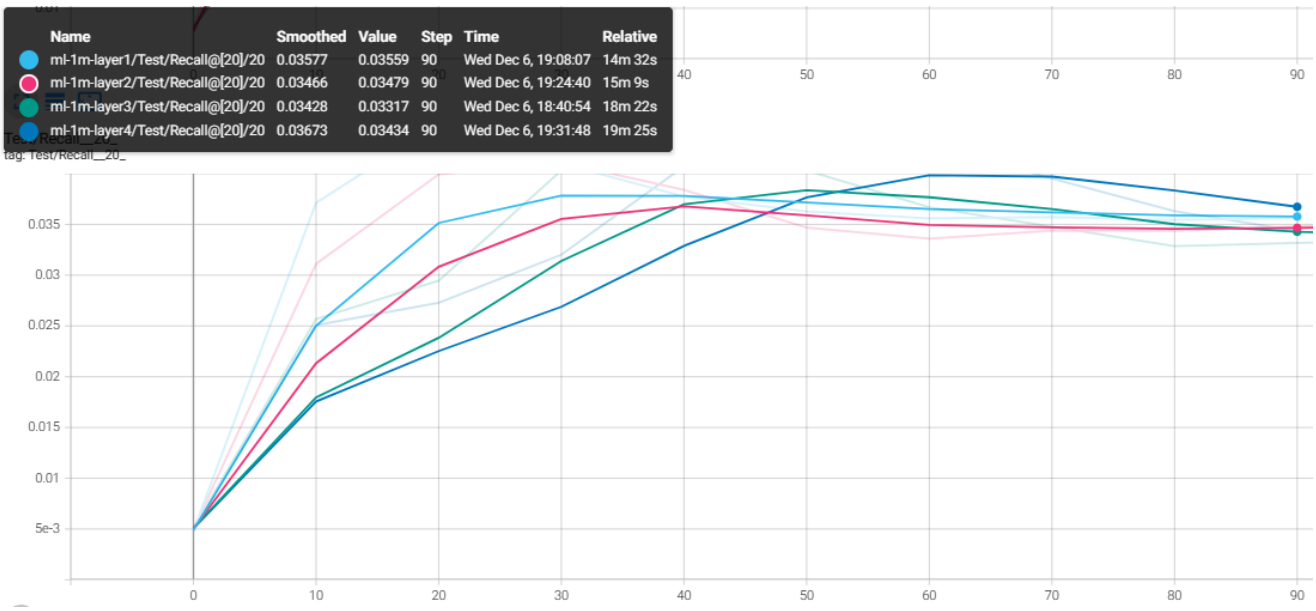
- ml_1m_NDCG



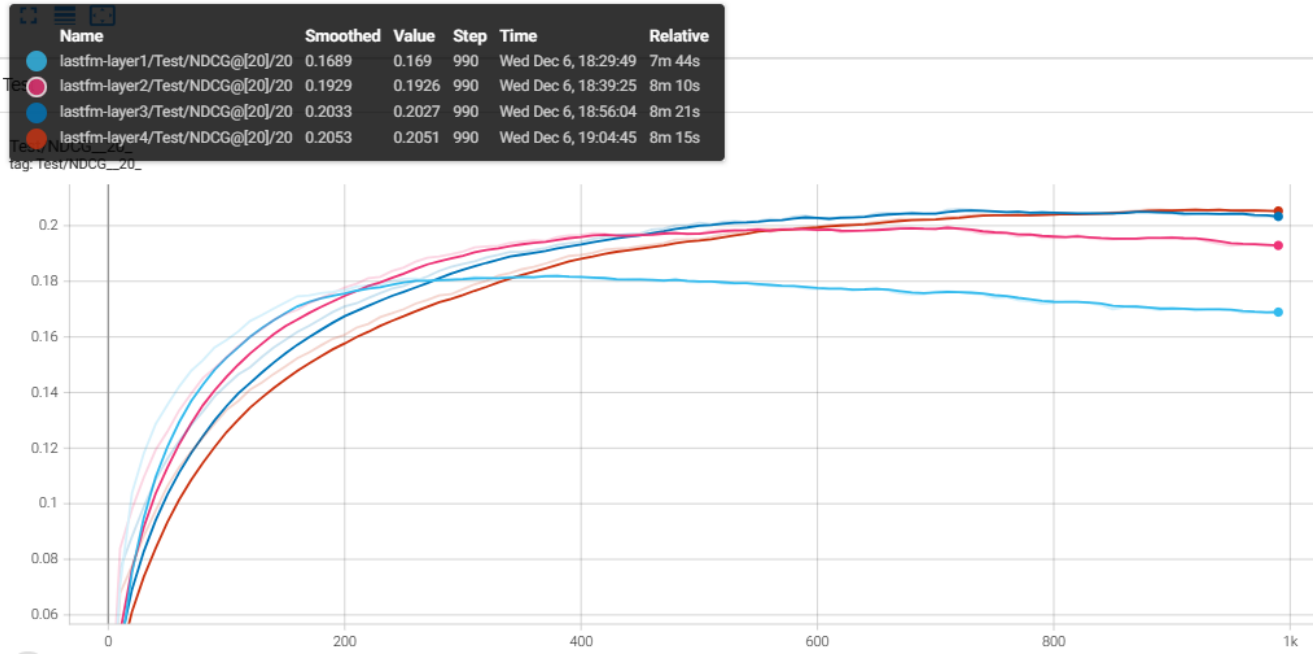
- ml_1m_Precision



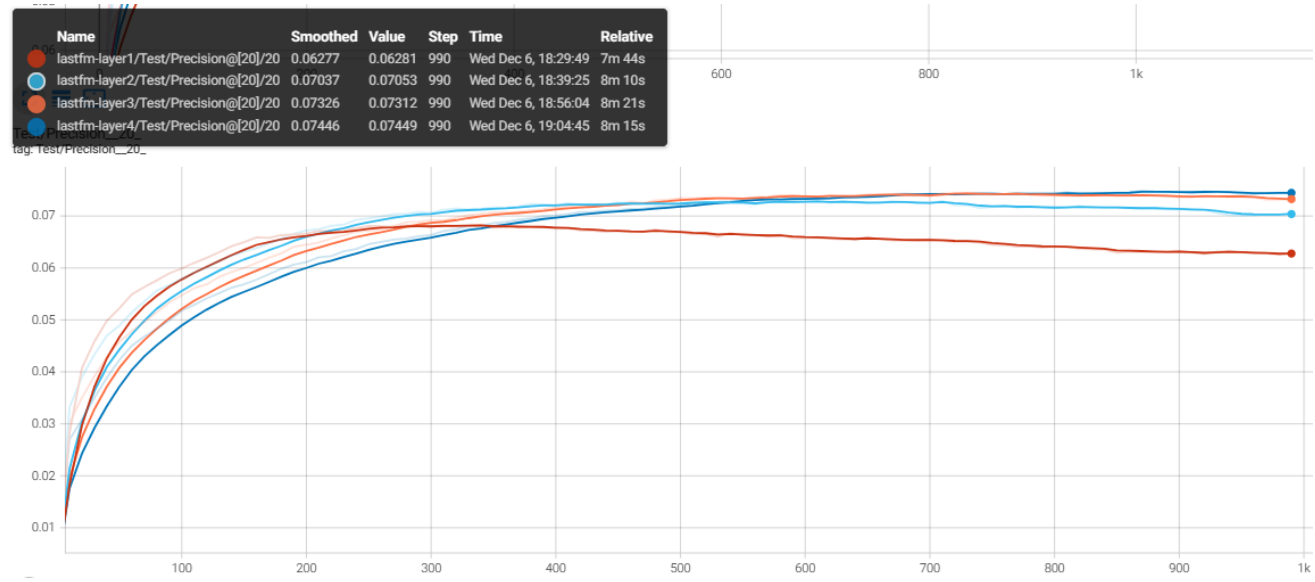
- ml_1m_Recall



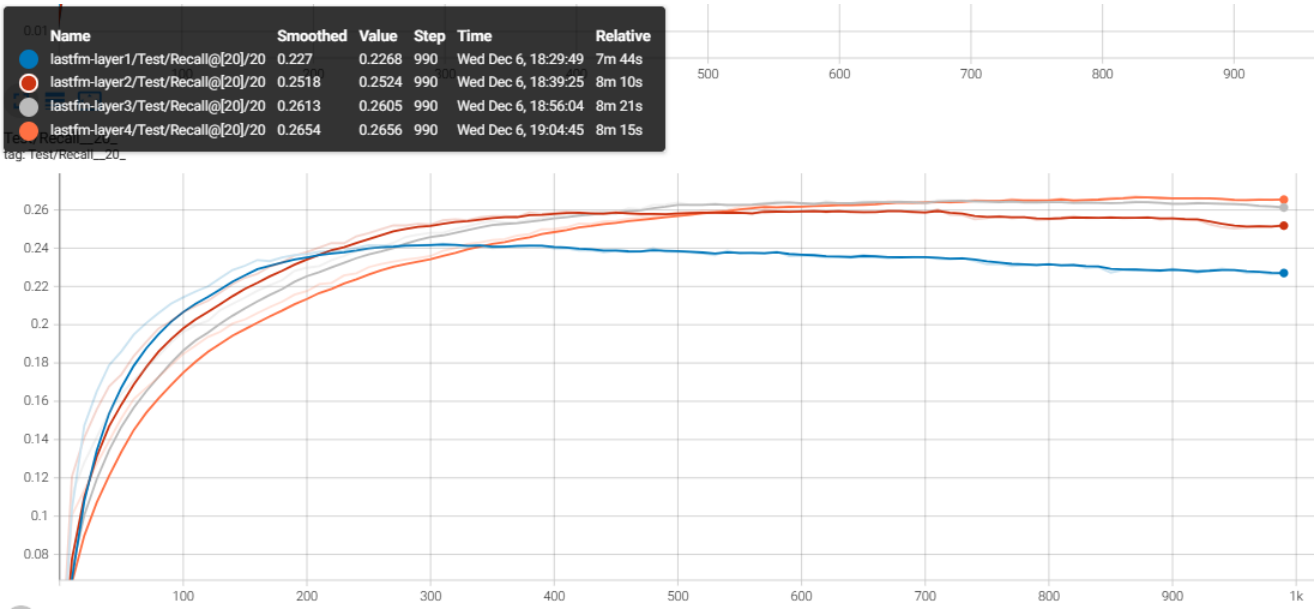
- hetrec2011-lastfm-2k_NDCG



- hetrec2011-lastfm-2k_Precision



- hetrec2011-lastfm-2k_Recall



Summary

Due to limitations in my personal computer's memory, the tests were conducted at the maximum scale possible.

From the charts, it's evident that increasing the number of layers significantly enhances the model's accuracy in predicting the top 20 recommendations. LightGCN performs well on the lastfm dataset, but its performance on ml-1m is subpar. Interestingly, at 50 epochs, it outperforms the model at 100 epochs. This could be due to the dataset's size, where the model's simplicity leads to over-smoothing.

Conclusion

On one hand, neural network algorithms have shown extraordinary potential in the field of recommendation systems, enabling better performance in more complex situations. However, neural networks needs a lot of data for training to get higher performance.

On the other hand, knowledge graphs provide rich semantic information that can help recommendation systems better understand the relationships between users and items. Combining knowledge graphs with recommendation systems can improve the accuracy and personalization of recommendations. It can alleviate the problem of insufficient data.

Inference

[2002.02126] LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation (arxiv.org)

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In Proceedings of the 26th International Conference on World Wide Web (WWW '17). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 173–182. https://doi.org/10.1145/3038912.3052569

Hailun Lin, Yong Liu, Weiping Wang, Yinliang Yue, Zheng Lin. Learning Entity and Relation Embeddings for Knowledge Resolution. Procedia Computer Science, Volume 108, 2017, Pages 345-354, ISSN 1877-0509.

Han, Xu and Cao, Shulin and Lv Xin and Lin, Yankai and Liu, Zhiyuan and Sun, Maosong and Li, Juanzi. OpenKE: An Open Toolkit for Knowledge Embedding. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations.

Appendix

Contribution

In order to efficiently complete the project, we divided the tasks and cooperated to achieve the final result

Name	**report**	**code**	**presenation**
Tong Shenyang	typical algorithm and GCN	typical algorithm and GCN	typical algorithm and GCN
Wang Pengxin	TransR	TransR	TransR

Name	report	code	presenation
Zhan Xiangcheng	NCF algorithms	NCF algorithms	NCF algorithms