Project Report : CS 7643

Shahrzad Amoozgar
Georgia Institute of Technology
samoozegar3@gatech.edu

Vida Gholami
Georgia Institute of Technology
vgholami@gatech.edu

Amin Mohammadbagheri
Georgia Institute of Technology
amin.mb@gatech.edu

Abstract

Recommender systems have been extensively used to analyze user experience patterns and suggest potential targets to users and customers in order to provide a more personalized experience. Recently Graph Neural Networks (GNNs) have become the state-of-the-art approach of recommender systems. The basic idea behind GNNs consists of using a neural network (NN) to obtain a representation for each node by aggregating the representations of neighboring nodes in a recursive fashion, limited to a certain depth of the network. These representations are used to predict the probability of potential links between nodes in a graph.In this work, we used GNN in order to recommend movies to users. We experimented with multiple methods for calculating similarities between users and movies, as well as different loss functions to train the model and predict potential links between users and movies.

1. Introduction/Background/Motivation

Our objective in this project was to build a model that is able to recommend movies to users. There are multiple approaches for this problem, however the state of the art approach is GNN. We used GNN and tried to leverage both user/movie features and the interactions between them in order to suggest new recommendations.

The traditional recommender systems like Matrix Factorization [13] capture the collaborative filtering effect between users and items by directly calculating the similarity of interactions. However these methods cannot capture the complexity of user behaviors. They do need the user-items interaction to work, as such can not be used in situations which are referred to as 'cold-start', like when a new movie is released. The other downside of these methods is the relevance of the recommendation, which only the popular items would get recommended. To address these issue, NN-based models are proposed [6].Thanks to the flexibility of the input layer of the network, specific features of users and items can be captured and recommendation can be more relevant. Neural collaborative filtering (NCF) and Deep Factorization Machine(DeepFM) [10] are two examples of NN based recommendation models. One of the issues with these types of recommender systems is ignoring high-order structural information in observed data. To address this, GNNs have been developed that adopt embedding propagation to aggregate neighborhood embedding iteratively. There are two types of Graph NN models: spatial and spectral models. GCN [12] and HGNN [8] are spectral GNN models which consider graphs as signals and process them with graph convolution in the spectral domain. GAT [15], GraphSAGE [11], and HetGNN [17] are different types of spatial models that spatially conduct the convolution on graph structures directly to extract localized features via weighted aggregation like CNNs [9].

Recommender systems play an important rule in e-commerce services, online audio or video-sharing platforms as well as media-services and web-content providers. Using recommender systems, companies can focus on increasing sales as a result of very personalized offers and an enhanced customer experience.This project is based on a research dataset, however the methods and experiments used for this project could be applicable to develop any similar recommender system.

2. Approach

As explained before, we used GNN modeling approach to create a movie recommendation system. There are different methods for developing recommendation systems, but our problem fits the definition of a graph. We have users and movies. We have their history of watched movies which can be used as a link, and our goal is to predict future links between users and movies. This approach has been utilized by a group of researchers at Uber Eats [5] to recommend restaurants to customers based on their history of interactions. Due to this similarity, we have used the same approach to recommend movies to users.

We used Deep Graph Library (DGL) which is a Python package built for implementation of GNN models, on top of existing Deep Learning frameworks (PyTorch and TensorFlow). The movieLens dataset used for this project, contains users' features(gender), and movies' features(genre) along with their interactions and the ratings that users assigned to each movie. An adjacency matrix is constructed based on users and movies interactions, this matrix is used to construct the schema for DGL graph. The ratings are formulated as edge weights. This recommender system would be successful for movie and music recommendation since it can leverage both content information (user, movie features) as well as graph structure (interactions between users and movies) to recommend movies to users.

We approached this problem by first processing the data. All the features related to users and movies are encoded through one-hot encoding. This was followed by creating the adjacency matrix carrying the connection information between each user and movie.For GNN-based collaborative filtering, the user-item bipartite graph is constructed according to users' interactions with items. We leveraged the capability of GNN and assigned features to users, and items nodes and edges. GNN is employed as an encoder to learn the representation of users and items through neural message passing.Each node feature is embedded and during each message passing iteration, the embedding $\boldsymbol{h}_u^{(k)}$ is updated using the aggregated information of neighborhood of node u, as shown in the following equation.

$$\boldsymbol{h}_u^{(k)} = \sigma\left(\boldsymbol{W}_{self}^{k}\boldsymbol{h}_u^{(k-1)} + \boldsymbol{W}_{neighbour}^{(k)} \sum_{v\in N(u)} \boldsymbol{h}_v^{(k-1)} + \boldsymbol{b}^{(k)}\right) \tag{1}$$

Where σ represents an element-wise non-linearity such as tanh or ReLU, $b^k \in \mathbb{R}^{d^{(k)}}$ denotes the bias term. $\boldsymbol{W}_{self}^{k}$, $\boldsymbol{W}_{neighbour}^{(k)} \in \mathbb{R}^{d^{(k)} \times d^{k-1}}$ are parameters matrices which we can train. u is the source node and v is the destination node.

The number of layers (k) defined, sets the k-hop neighborhood of the node. After k iterations of GNN message passing, the embedding of each node might encode information about all the nodes in that node's k-hop neighborhood. These node embeddings were then compared against each other using some similarity calculation functions and a rating was assigned to each. Based on the number of recommendations required the top ranked movies for each user was retrieved as the outcome.More details are provided in the modeling section.

3. Dataset

We used "MovieLens 1M" dataset[3] from GroupLens Research Project which is a research group in the Department of Computer Science and Engineering at the University of Minnesota.This dataset contains user rating data for movies collected from 2000s and includes three sets of data as follows:

- Movie ratings: 1,000,209 anonymous ratings (5-star scale) of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000.
- Movies metadata such as title, genre and production year.
- Demographic information of the users such as age, zip code, gender identification and users occupation.

Why Movies? Recommendation systems changed the way commercial websites communicate with their users such as Amazon, Netflix, Google, etc. They increase interactions between the users and their product to provide a better experience. As more people are forced to stay at home or work from home, building powerful movie recommender systems bring benefits for both users and streaming providers. Movielense 1M dataset is a stable large dataset which contains user demographic information, thus it helps us to integrate user gender.

3.1. Pre-processing:

In order to prepare the data for GNN, We used one-hot encoding to convert categorical features of users and movies into binary variables.The only features used for movies, and users is genre of the movie and the gender of the user. These feature vectors are fed into DGL graph which is constructed from the adjacency matrix representing user and item interactions. This adjacency matrix is a sparse binary matrix which would be converted into a DGL graph. Then DGL graph structure which stores features of nodes(movies and users) and weights of edges(rating of movies), would be used to train the model.

4. Model Architecture

Upon building the graph, the information for each node was embedded to a new dimension of 256. These embedded information were then passed through our GNN model.However since the dataset is large, we had to divide it into mini-batches. Something that was important to be considered in a link prediction GNN was "negative sampling". We defined a negative sampler that randomly chooses negative destination nodes for each source node in the mini-batch according to a uniform distribution, and creates a new link.The number of negative edges is one of the

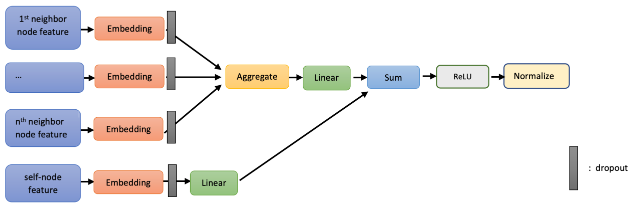


Figure 1: The GNN Layer Architecture – At each layer the node and neighboring node features get embedded and go through an aggregation process, the updated value is assigned as the node features in the next layer.

hyper-parametrs for this model. For each mini-batch, a positive graph, negative graph and message flow graph (MFG) is defined. The MFG contains as many blocks as the number of layers (neighborhood hops) which is defined as a hyper-parameter, and in each block it contains the message passing information from the neighboring nodes to each node. For example, a 'full' sampler, with 2 layers would propagate messages through all the immediate neighboring nodes in layer 1 (Block 1), and all the 2-hop neighboring nodes in layer 2 (Block 2). These message passing connections are called MFGs. As the dataset is large, having full neighborhood comes with a computation price, hence a "partial" sampler can be used with adjustable number of nodes in each layer for building the message passing flow diagram. Using a convolution layer, at each block, the MFG is used to pass the embedded information (feature) from the neighboring node to the individual nodes in that layer, and after aggregation, the node features are updated.The new node features are then passed to the next layer through the MFGs of that layer and the update process continues for as many layers as we set before. After completing message passing, a similarity metric is calculated between the updated user node feature and updated movie node feature. This is later used for performing the link prediction and recommendation.

5. Optimization

After producing graph structure and the embedding of nodes, these embeddings would be used to generate scores for edges(e.g. the probability that an edge exists) for the downstream link prediction task. The relevant embeddings and the labels would be utilized to formulate the loss function, and the existing optimizers are used for model learning. We have used several mapping or similarity functions(e.g.MLP, Cosine similarity) to generate scores and several loss functions to train the best model.

5.1. Similarity Measure

After the input layer and GNN layer, there is a component called task layer. Its functionality is essentially computing task-dependent output which will be used in the loss function, while training the network. When we want to compare two nodes or entities such as users and movies, we need to use the feature embedding. The embedding typically maps the source feature data to an embedding space with fewer dimensions, in a way that captures the latent structure of the source dataset. The embedding can handle redundant features and correlation between them, and the similar entities become closer in an embedding space.The similarity measure takes the embedding of two element in a pair and returns a similarity score. There are multiple methods to calculate the similarity measure. Below is a brief explanation of each of the methods we implemented and tested.

5.1.1 DotProduct Similarity

One of the most important application of dot product is measuring the similarity between feature vectors. The samples that are frequent in the training set (most watched movies) will have feature embedding vectors with larger length. This method is useful in measuring similarities since the dot product is proportional to vector length, and the use of dot product is recommended if we want to consider the popularity of movies as well.

5.1.2 Cosine Similarity

The potential risk with using dot product is that popular items might skew the similarity. One way to balance this is to normalize vectors. Cosine similarity is essentially the dot product scaled by magnitude, and because of scaling it is normalized between 0 and 1 [4].

5.1.3 Neural Network Predicted Similarity

Another approach is using general NNs to produce a scalar score on each edge by concatenating the incident nodes' features (user and movie) and passing it to an MLP, according to Figure 2. Figure 3, shows the structure of the MLP layer we used for edge score computation. We passed the embedding through multiple linear layers, first increasing the hidden dimension to 256, to capture as much information from embedded features as possible, then decreasing it back to 1 to get a score. Each linear layer was followed by a ReLU activation function to add non-linearity [7].

5.2. Loss Function

Another key component for recommender systems is to define a proper objective function for model optimization

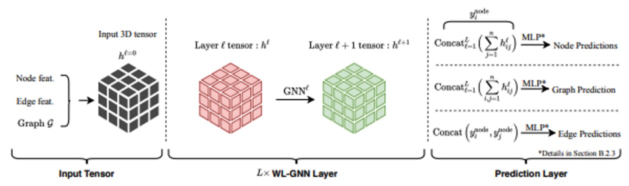

Input 3D tensor
$h^{\ell=0}$
Node feat.
Edge feat.
Graph $\mathcal{G}$
Layer ℓ tensor : h^{ℓ}
GNN^{ℓ}
Layer $\ell+1$ tensor : $h^{\ell+1}$
y_i^{node}
$\text{Concat}_{\ell=1}^{L}\left(\sum_{j=1}^{n} h_{ij}^{\ell}\right) \xrightarrow{\text{MLP*}}$ Node Predictions
$\text{Concat}_{\ell=1}^{L}\left(\sum_{i,j=1}^{n} h_{ij}^{\ell}\right) \xrightarrow{\text{MLP*}}$ Graph Prediction
$\text{Concat}\left(y_i^{\text{node}}, y_j^{\text{node}}\right) \xrightarrow{\text{MLP*}}$ Edge Predictions
*Details in Section B.2.3
Input Tensor
$L\times$ WL-GNN Layer
Prediction Layer

Figure 2: A standard experimental pipeline for Weisfeiler-Lehman-GNNs, which inputs to a GNN a graph with all node and edge information (if available) represented by a dense tensor, performs several GNN layer computations over the dense tensor, and finally makes a prediction through a task-specific MLP layer [7]

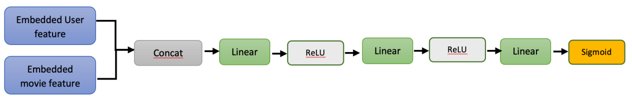


Figure 3: The MLP predictor Architecture – The embedded user and movie features go through a deep network in order to produce similarity measure.

according to the observed data and unobserved interactions. Point-wise and pair-wise objective functions are two commonly used objective functions in recommender systems.In this work we used three different loss functions which include both point-wise and pair-wise loss functions.

While calculating the loss, we considered a new tensor, called ”negative mask”. As noted in the approach, negative edges were built through a sampling process, which would uniformly connect the node to some other nodes in the graph and create new links which form the negative graph. However, these created edges might be false negative edges as they already exist. To prevent counting for them all the pre-existing edges were removed from negative edges using ”negative mask”.

5.2.1 Max Margin Loss

The basic idea of max-margin based loss function is to maximize the similarity score of positive examples and minimize the similarity score of negative examples.[16] Max-Margin loss can be calculated as:

$$J_G(z_q z_i) = E_{n_k \sim P_n(q)} max\{0, z_q.z_{n_k} - z_q.z_i + \Delta\} \quad (2)$$

where $P_n(q)$ represents the distribution of negative examples for item q, and Δ denotes the margin hyper-parameter. If Δ is too high, the task would be very difficult for the model to learn, and if it is too low, the learning performance would diminish due to low scores given to negative pairs.

5.2.2 Bayesian Personalized Ranking(BPR) Loss

BPR loss [14] is a pairwise loss function used in the area of recommender systems. BPR assumes that observed interactions that indicates user preferences should be assigned higher prediction values than unobserved interactions. Through this loss function, BPR essentially attempts to discriminate between positive and negative samples.

$$L_{CF} = \sum_{(u,i,j)\in O} - \ln \sigma\Big(\hat{y}(u,i) - \hat{y}(u,j)\Big) \quad (3)$$

where $O = (u,i,j)|(u,i) \in \mathbb{R}^+, (u,j) \in \mathbb{R}^-$ denotes the training set, $\mathbb{R}^+$ represents the observed positive interactions between user u and movie j while $\mathbb{R}^-$ indicates the sampled unobserved (negative) interaction set, and $\sigma(.)$ is the Sigmoid function.

5.2.3 Cross Entropy Loss

Cross entropy loss is used to compare how well the probability distribution output by model matches the ground truth label of the data. The following equation calculates the log loss for a single sample with true label $y \in 0,1$:

$$CE(y,p) = -(ylog(p) + (1-y)log(1-p)) \quad (4)$$

where p denotes the probability estimate as $p = Pr(y = 1)$

6. Experiments and Results

We experimented with different similarity functions including cosine similarity, dot product and using a general NN for predicting the similarity which is used for ranking the recommended movies. Our experiments showed that using NN (MLP layer) for calculating the similarity as a scalar score was the most effective method (Table 1). This is because we are comparing the embedded features (tensors), and NN is capable of capturing more information which can be reflected in the similarity score, in contrast to the simple vector projection performed in either dot-product or cosine similarity computation.

Loss calculation is another important field we focused on. We experimented with Max Margin Loss, BPR and Cross Entropy loss (for classifying a link as a true or false connection). Based on the results of our experiments Max Margin is better suited to our problem, since it achieved higher precision, and AUC (Table 2). Although BPR and Max margin are both pair-wise loss functions and very similar in nature, Max Margin performed slightly better than BPR. It seems that changing safety margin inside Max Margin formula was one of the contributing factors in having better results.

The results of the best model are shown in Figure 4 and Figure 5 . The results for the above-mentioned experiments are brought in Appendix Experiment Results and HP Table.

6.1. Model Evaluation Metric

The core part of building a reliable model is evaluation. To compare different models we obtained in the hyper-parameter tuning section, different metrics were used, to evaluate the best model for inference.

Loss function behavior on both training and validation set was a major metric in our evaluation. It is understandable that validation loss is generally higher than training loss, however since our optimizer should minimize the objective function, we expect a decreasing trend for both training and validation set. If at a point, the training loss continues to decrease but validation loss starts an increasing trend, this would be an indicator of over-fitting, meaning the model has memorized the training data, and does not have generalization capability. We are using an early stopping criteria "patience", which would stop the model if the validation loss does not improve after predetermined number of epochs.This would prevent the model to over-fit.

Precision and recall are two commonly used metrics when it comes to classification modeling. Link prediction can be considered as a binary classification for which positive class represents the observed edges and negative class represents the unobserved links. In this problem, we are more interested in true positives(correct recommendations), and false-negatives are less of a concern. Precision and recall can be calculated based on the formulas shown in Equations (Equations 5 and 6). High values of precision suggests high probability of retrieving relevant results, and high values of recall means that the algorithm can retrieve most of the relevant results. For the explained reasons, we considered precision as the evaluation metric.

AUC is the area under receiver operating Characteristic curve(ROC).AUC can be used to measure the ability of a classifier to distinguish between classes. The AUC-ROC curve is generated by changing the discriminator classifier threshold (usually defaulted at 50 percent) and plotting the true positive rate TPR) against the false positive rate (FPR) (Equations 7 and 8). AUC equals to 0.5 when the ROC curve corresponds to random chance and 1.0 for perfect classifier.

6.2. Hyper-parameter Tuning

Hyper-parameters are the variables that govern the training process. Table 4 in the Appendix shows the number of hyper-parameters we had for this model. As shown in the table, for each parameter a configuration list was defined and the hyper-parameters were tuned based on the results of model. The evaluation metrics explained in the previous section were used to evaluate the performance of the models and attain the hyper-parameters that would have the best/most reasonable results, which do not show over-fitting and at the same time, yield the highest AUC and precision value.

While some of the hyper-parameters were significantly affecting the performance of the model, some were not as important.

The following parameters were more important in this study. A brief explanation has been provided for each:

Effect of number of layers in Neural Network predicted similarity: We can add as many linear and nonlinear layers to NN similarity function, however too many layers would result in over-fitting because it would make the network memorize all the training data, and causes poor performance on validation and test data, and too few would result in underfitting and not learning the data.

Effect of hidden layers dimensionality in NN predicted similarity: We investigated how the performance changes with respect to the dimension of the hidden layers (number of units) in NN similarity layer. We noticed the performance first increases and then begins to decrease when the fist hidden dimension size reaches 256 and the second hidden dimension reaches 128, which indicates these dimensions represented enough information and the model will be overfitted with higher number of units.

Effect of number of GNN's layers: Even though some studies focus on increasing GNN's layers to capture higher-order connectivity relations on graphs to improve models' performance,our experiment results show that partial nodes require just two layers of propagation to boost model performance, and that was the sweet spot for neighborhood message passing.

Effect of negative sample size: the negative sampling of recommender systems has an important influence on the training of the model. Too few samples lead to poor learning, and too many would make the data very imbalanced and slow down the training without improving the performances.We achieved the best results for negative sample size of 15.

Effect of train/validation size: We experimented with different sizes of train and validation set. If validation set is too small, it does not provide sufficient information to evaluate the ability of the model to generalize, and if it is too large then the training set would not have sufficient data to train a reasonable model. The optimum size of validation set was 15% for training this model.

7. Challenges

One of the anticipated problems was the scarcity of available resources for using the new framework to implement GNNs(DGL).Even though the library was introduced in 2019, there were lots of informative documents and examples provided in DGL website. However it was after we

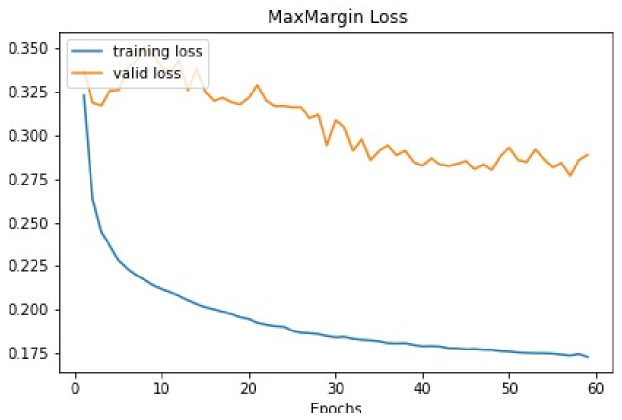


Figure 4: Loss Function - MAX Margin Loss and NN Similarity Pred

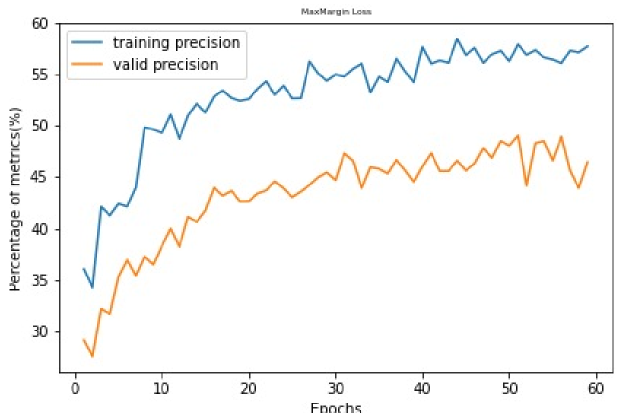


Figure 5: Precision Metric - MAX Margin Loss and NN Similarity Pred

Similarity Function	Precision on Validation Set	AUC on Validation Set
NN Predicted Similarity	45.5%	77.4%
Cosine Similarity	30.9%	77.4%
Dot Product Similarity	28.2%	76.9%%

Table 1: Experiment 1 - Results at Epoch 30

started working, that we realized the available information does not address many of questions we faced during implementation of the project. Although there are many sample codes available for recommender systems, they rarely use GNN, and the available related materials are at an abstract level. One of the unique aspects of our project was modeling a heterogeneous graph (a type of graph with different nodes and edge types),and performing link prediction as opposed to a homogeneous graph and classification task, which in turn played a role in not having enough previous implementations.

Loss function	Precision on Val. set	AUC on Val. set
Max Margin loss	45.5%	77.4%
BPR	37.4%	73.8%
Cross Entropy	2.1%	48.9%

Table 2: Experiment 2 - Results at Epoch 30

The main challenge in our work, which we had not anticipated to this extent, was the amount of hyper-parameters that we had to tune. This made the model training extremely challenging, and improving the performance of the model extremely difficult given the limited time we had to work on this project.

Due to the novelty of this approach and the type of data we used in this project(graph) compared to image and text data, we had to study different materials and use previous implementations, and adapt them to our specific type of problem.

The main source code that was used to develop our model, was based on the implementation of a recommender system to suggest items to users developed by researchers at a Canadian sporting goods retailer [2]. We also frequently used DGL library documents and examples [1] to learn and develop our model. Tailoring these implementations to our specific problem, and finding the best structure of the model, optimization method and loss function as well as hyper-parameter tuning was a very challenging task.

8. Future work

In this work, we developed a recommender system that can recommend movies to users based on their historical interactions and individual features. However, we did not address the cold start problem in our recommender system, a condition in which the system should recommend items to new users or make recommendation for new items that had no or little history of interactions. Since GNNs heavily rely on user-item interactions, this can be challenging for our model to solve such problems. In future, we will develop a hybrid recommender system that can address the cold start problem by utilizing users and items features, this would require collecting more data regarding new users and items.

References

[1] Deep graph library. https://docs.dgl.ai/en/0.6.x/. 6

Student Name	Contributed Aspects	Details
Amin M.	Data Pre-processing	Analyzed and converted the data into a graph structure
Amin M., Shahrzad A.,Vida Gh.	Model architecture implementation	Designed and implemented the message passing functionality for GNN
Vida Gh.	Similarity functions implementation	created and experimented with different similarity functions including MLP, Cosine similarity and dot product
Shahrzad A.	Loss functions implementation	Created and experimented with different loss functions including max-margin, cross entropy, and BPR
Amin M., Shahrzad A.,Vida Gh.	Model training	Designed and implemented the complete pipeline for training and evaluating the model
Amin M., Shahrzad A.,Vida Gh.	Hyper-parameter tuning	Experimented with different hyper-parameters to improve the performance of the model
Amin M., Shahrzad A.,Vida Gh.	Writing report	Literature search and report writing

Table 3: Contributions of team members.

[2] Gnn-recsys. https://github.com/je-dbl/GNN-RecSys. 6

[3] Grouplens. https://grouplens.org/datasets/movielens/1m/. 2

[4] Measuring similarity from embeddings. https://developers.google.com/machine-learning/clustering/similarity/measuring-similarity. 3

[5] Ankur Sarda Ankit Jain, Isaac Liu and Piero Molino. Food discovery with uber eats: Using graph learning to power recommendations, 2019. 1

[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide and deep learning for recommender systems, 2016. 1

[7] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks, 2020. 3, 4

[8] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3558–3565, Jul. 2019. 1

[9] Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, and Yong Li. Graph neural networks for recommender systems: Challenges, methods, and directions, 2021. 1

[10] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for CTR prediction. *CoRR*, abs/1703.04247, 2017. 1

[11] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc. 1

[12] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. 1

[13] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. 1

[14] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback, 2012. 4

[15] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018. 1

[16] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, Jul 2018. 4

[17] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '19, page 793–803, New York, NY, USA, 2019. Association for Computing Machinery. 1

A. Appendix

A.1. Evaluation Equations

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

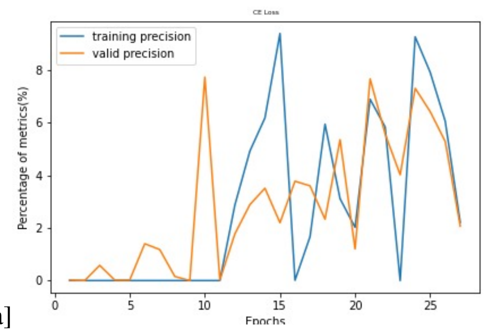
$$TPR = \frac{TP}{ActualPositive} = \frac{TP}{TP + FN} \tag{7}$$

$$FPR = \frac{FP}{ActualNegative} = \frac{FP}{TN + FP} \tag{8}$$

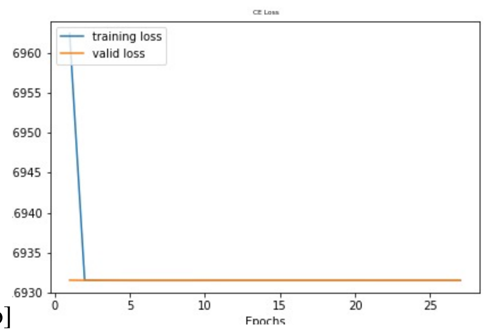
A.2. Experiment Results and HP Table

Hyper-parameter	Config List	Tuned Parameter
Num. of epochs	[10, 30, 60]	60
patience for early stopping		10
Num of recommendations		5
Num. of GNN layers	[2, 3, 4, 5]	3
Neighbor sampler	['full','partial']	'partial'
Samples per block		[50,40]
Neg. Sample Size	[5, 10, 15, 50]	15
Eedge batch size		4096
node batch size		128
loss func	[Max Margin Loss, BPR loss, CE loss]	max-margin loss
GNN layer hps		
out dimension	[32,64,128,256]	64
hidden dimension	[64,128,256,512]	256
similarity predictor	['cos','nn','dotprod']	'nn'
aggregator	['mean', 'sum', 'max']	'mean'
dropout	[0.1, 0.2,0.3,0.5]	0.3
similarity Pred hps		
NN-hid.layer1 out dim	[32,64,128,256]	256
NN-hid.layer1 out dim	[32,64,128,256]	128
MaxMargin delta	[0,0.1,0.3,0.6,1]	0.6
Optimizer hps		
Opt. Algorithm	[SGD,Adam,AdamW]	Adam
learning rate	[1e-5, 1e-3, 1e-1]	1e-1
weight decay	[1e-3,1e-4,1e-5]	1e-5

Table 4: List of Hyper-Parameters and Tuning Result

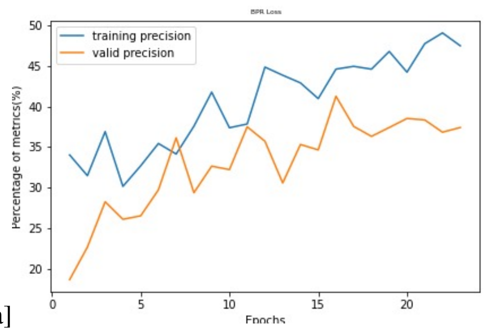


[a]

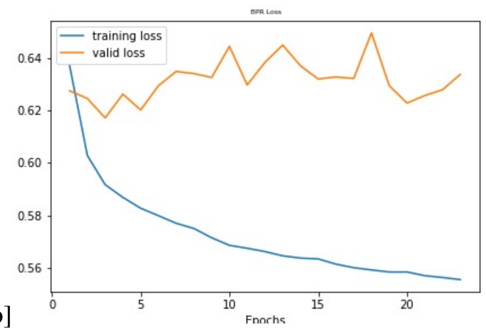


[b]

Figure 6: (a) Precision-Metric-CE Loss-NN Similarity Pred (b) Loss-Function-CE Loss-NN Similarity Pred

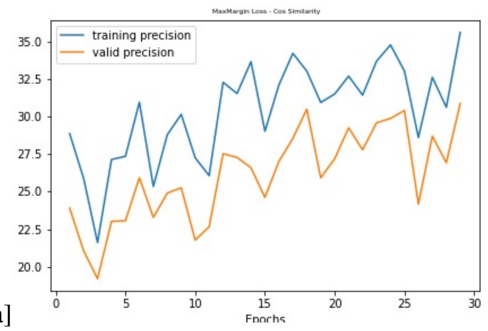


[a]

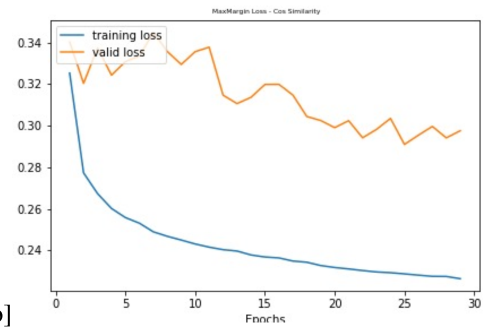


[b]

Figure 7: (a) Precision Metric - BPR and NN Similarity Pred (b) Loss Function - BPR Loss and NN Similarity Pred

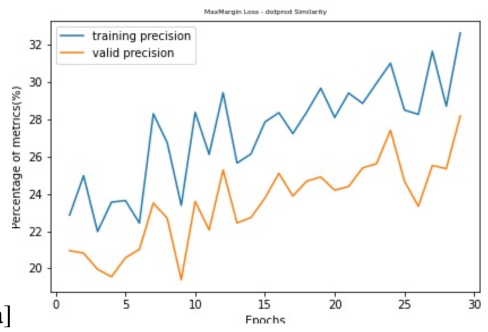


[a]

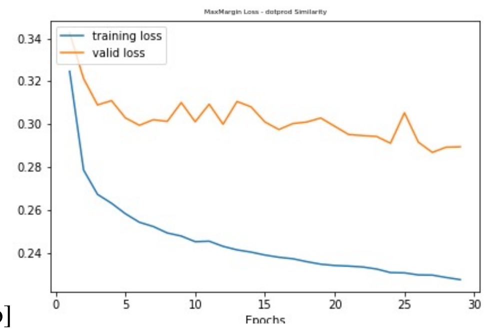


[b]

Figure 8: (a) Precision Metric - Cosine Similarity Pred and Max Margin Loss (b) Loss Function - Caption: Loss Function - Cosine Similarity Pred and Max Margin Loss



[a]



[b]

Figure 9: (a) Precision Metric - Dot Prod Pred and Max Margin Loss (b) Loss Function - Cosine Similarity Pred and Max Margin Loss