GNN Optimization

A Report of Spring EE595 Project Submitted by
by

Yi Deng
9696485532
&
Muhao Guo
3441981541
&
Qian Wang
4116380919

Supervised by:
Mohammad Saeed Abrishami

Abstract

In this report, we briefly introduce what we have done for our GNN project by now. We first describe our problem, then the project timeline, which includes the papers we have read, the lecture nodes we have reviewed and the algorithms we have implemented. Then, we introduce the data set on which we use GNN optimization. Also, the algorithms we used are performed through pseudo-code and analyzed with flow chart. In the end, we sketched out our implementation and discussed the challenges we may come across.
Keywords: GNN optimization, GCN, PinSage, dgl

I Problem Description

Graph study is trending now due to the ability of representing the real world such as social networks, bioinformatics, molecular structures, circuit elements etc. And nowadays GNNs has become a powerful and popular tool for machine learning on graphs. The purpose of this project is to implement GNN optimization based on PyTorch to solve problems in social networks. More specifically, our job can be split into two parts. For same-typed nodes, like citations, we use GCN to do node embedding. And for different-typed nodes, like users and movies, we apply PinSage on Netflix prize dataset in order to predict how the users rate movies.

II Project Timeline

A Phase I

Papers we have read

- Semi-supervised Classification with Graph Convolutional Networks *by Thomas N. Kipf and Max Welling*
- DeepWalk: Online Learning of Social Representations *by Bryan Perozzi, Rami Al-Rfou and Steven Skiena*
- node2vec: Scalable Feature Learning for Networks *by Aditya Grover and Jure Leskovec*
- Skipgram: Distributed Representations of Words and Phrases *by Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean*

Lecture Nodes we have reviewed

- Stanford CS224W: Motifs and Structural Roles in Networks
- Stanford CS224W: Community Structure in Networks
- Stanford CS224W: Message Passing and Node Classification
- Stanford CS224W: Graph Representation Learning

Algorithms we have practiced or implemented

- Pytorch Transfer Learning practice
- Implemented DeepWalk Algorithm
- Implemented node2vec Algorithm
- Wrote SkipGram Algorithm in Pytorch

B Phase II

Papers we have read

- Graph Convolutional Neural Networks for Web-scale Recommender Systems *by Rex Ying, Ruining He, Kaifeng Chen, etc.*

Lecture Nodes we have reviewed

- Stanford CS224W: Graph Neural Networks
- Stanford CS224W: Graph Neural Networks: Hands-on Session
- Stanford CS224W: Message Passing and Node Classification
- Stanford CS224W: Machine Learning with Graphs (Video)

Implementation

- implement GCN in Citation Full dataset to do node classification

C Phase III

Papers we have read

- Advancing GraphSAGE with A Data-Driven Node Sampling *by Jihun Oh, Kyunghyun Cho, Joan Bruna*
- Graph Convolutional Neural Networks for Web-scale Recommender Systems *by Rex Ying, Ruining He, Kaifeng Chen, etc.*
- EDGNN: A Simple And Powerful GNN For Directed Labeled Graphs *by Guillaume Jaume and An-phi Nguyenz*

implementation

- Implement pinsage in Netflix Dataset for movie recommendation.

III GCN

A Algorithm

The graph convolutional operator(GCN) from the "Semi-supervised Classification with Graph Convolutional Networks" paper[1]
A multi-layer Graph Convolutional Network (GCN) with the following layer-wise propagation rule:
$H^{(l+1)} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)})$
$\tilde{A} = A+I_N$ is the adjacency matrix of the undirected graph G with added self-connections. I_N is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ and $W^{(l)}$ is a layer-specific trainable weight matrix. $\sigma(\cdot)$denotes an activation function, such as the $ReLU(\cdot) = max(0, \cdot)$. $H^{(l)} \in R^{N\times D}$ is the matrix of activations in the l^{th} layer; $H^{(0)} = X$.

B Dataset

The full citation network datasets from the "Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning [2] (Dataset Download). Nodes represent documents and edges represent citation links. Datasets include 'citeseer', 'cora', 'cora_ml', 'dblp', 'pubmed'. Dataset is shown in Table 1.

dataset	num of edges	num of nodes	size of node features	num of classes
cora	126842	19793	8710	70
coraml	16316	2995	2879	7
citeseer	10674	4230	602	6
dblp	105734	17716	1639	4
pubmed	88648	19717	500	3

Table 1: Dataset Information

C Model Structure

The model structure we used is show in Figure 1.

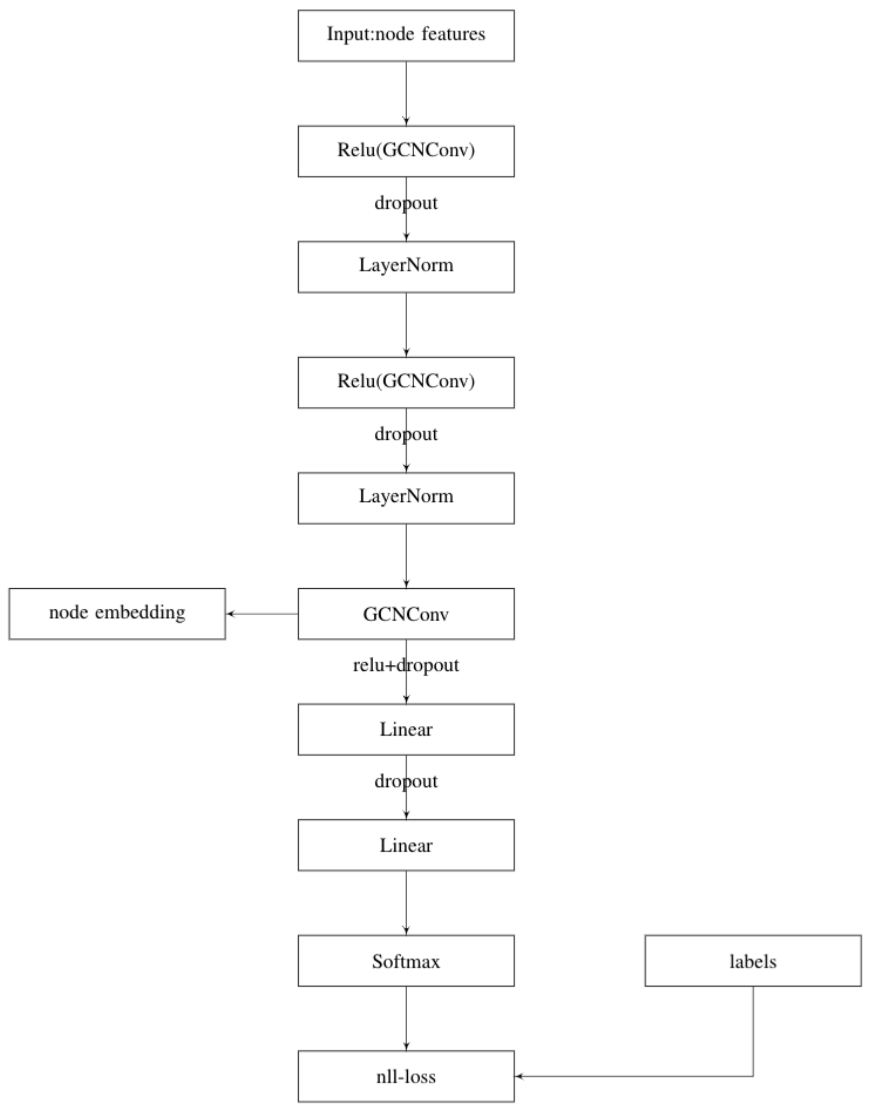


Figure 1: GNN Stack Model Structure

D Results

The result visualization on dataset Citeseer is in Figure 2.

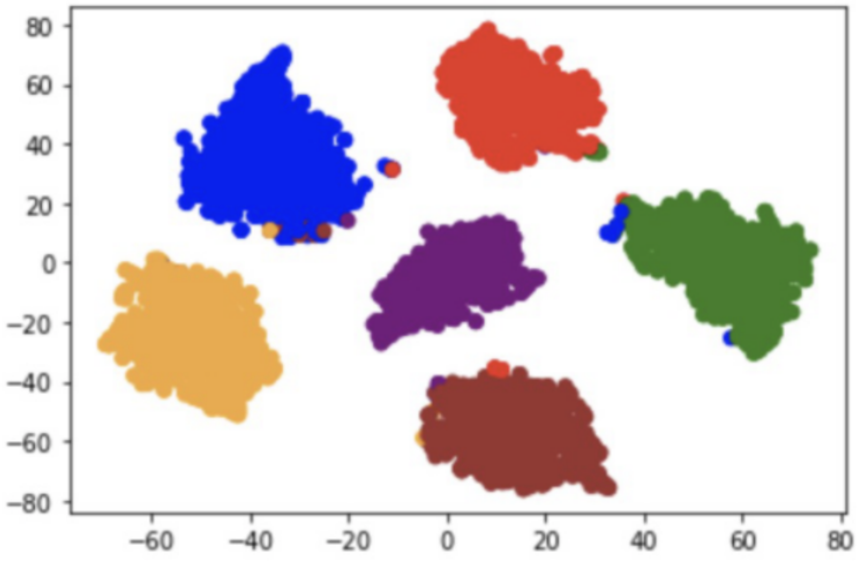


Figure 2: Result Visualization on Citeseer

The result visualization on dataset dblq is in Figure 3.

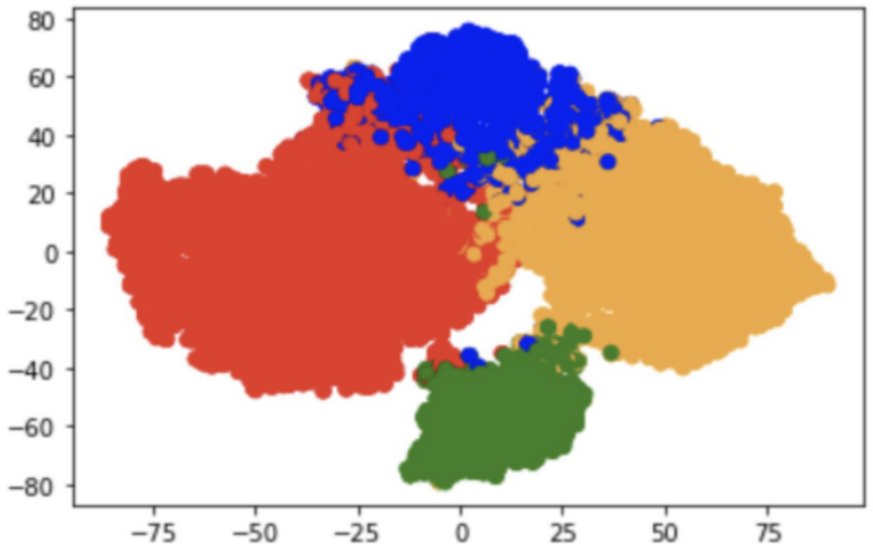


Figure 3: Result Visualization on Dblq

The result of each dataset is shown in Table 2.

dataset	cora	coraml	citeseer	dblq	pubmed
accuracy	0.8480	0.9790	0.9960	0.9434	0.9328

Table 2: Results on each dataset

E Result Analysis

The result shows that the classification performance is very good. As all of the classes is well separated. Also, from the result table, we know that the accuracy is relatively high, which suggests the performance is good.

IV Film Recommendation

A PinSage Algorithm

The core of our PinSage [3] Algorithm is a localized convolution operation, where we learn how to aggregate information from u's neighborhood. Pinsage Algorithm is shown in Figure 4. Left: A small example input graph. Right: The 2-layer neural network that computes the embedding hA2 of node A using the previous-layer representation,hA1, of node A and that of its neighborhood N (A) (nodes B,C,D)

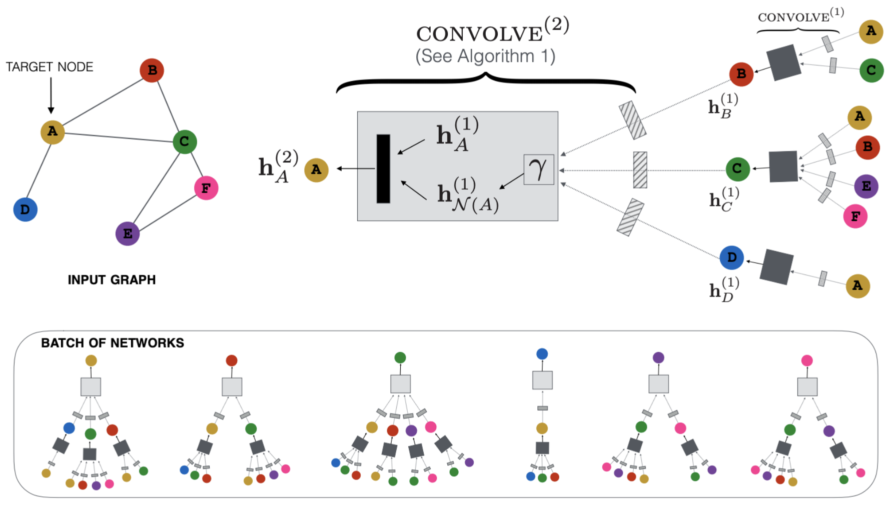


Figure 4: PinSage

The convolve part is shown in Figure 5

```
Algorithm 1: CONVOLVE
Input  : Current embedding $\mathbf{z}_u$ for node $u$; set of neighbor
         embeddings $\{\mathbf{z}_v | v \in \mathcal{N}(u)\}$, set of neighbor weights
         $\boldsymbol{\alpha}$; symmetric vector function $\gamma(\cdot)$
Output : New embedding $\mathbf{z}_u^{\text{NEW}}$ for node $u$
1 $\mathbf{n}_u \leftarrow \gamma\left(\{\text{ReLU}\left(\mathbf{Q}\mathbf{h}_v + \mathbf{q}\right) \mid v \in \mathcal{N}(u)\}, \boldsymbol{\alpha}\right)$;
2 $\mathbf{z}_u^{\text{NEW}} \leftarrow \text{ReLU}\left(\mathbf{W} \cdot \text{CONCAT}(\mathbf{z}_u, \mathbf{n}_u) + \mathbf{w}\right)$;
3 $\mathbf{z}_u^{\text{NEW}} \leftarrow \mathbf{z}_u^{\text{NEW}} / \|\mathbf{z}_u^{\text{NEW}}\|_2$
```

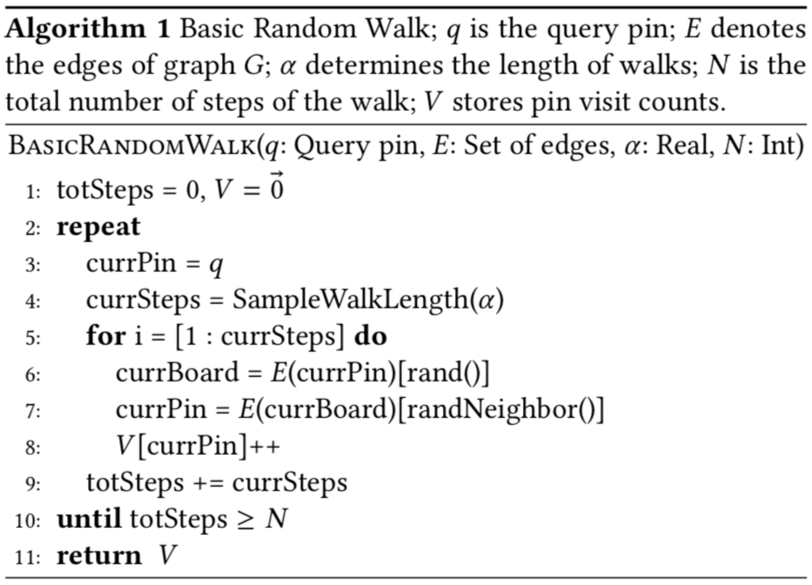
Figure 5: PinSage convolve

The minibatch part is shown in Figure 6

```
Algorithm 2: MINIBATCH
Input  : Set of nodes $\mathcal{M} \subset \mathcal{V}$; depth parameter $K$;
         neighborhood function $\mathcal{N} : \mathcal{V} \rightarrow 2^{\mathcal{V}}$
Output : Embeddings $\mathbf{z}_u, \forall u \in \mathcal{M}$
   /* Sampling neighborhoods of minibatch nodes.    */
1  $\mathcal{S}^{(K)} \leftarrow \mathcal{M}$;
2  for $k = K, ..., 1$ do
3      $\mathcal{S}^{(k-1)} \leftarrow \mathcal{S}^{(k)}$;
4      for $u \in \mathcal{S}^{(k)}$ do
5          $\mathcal{S}^{(k-1)} \leftarrow \mathcal{S}^{(k-1)} \cup \mathcal{N}(u)$;
6      end
7  end
   /* Generating embeddings                         */
8  $\mathbf{h}_u^{(0)} \leftarrow \mathbf{x}_u, \forall u \in \mathcal{S}^{(0)}$;
9  for $k = 1, ..., K$ do
10     for $u \in \mathcal{S}^{(k)}$ do
11         $\mathcal{H} \leftarrow \left\{\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u)\right\}$;
12         $\mathbf{h}_u^{(k)} \leftarrow \text{CONVOLVE}^{(k)}\left(\mathbf{h}_u^{(k-1)}, \mathcal{H}\right)$
13     end
14 end
15 for $u \in \mathcal{M}$ do
16     $\mathbf{z}_u \leftarrow \mathbf{G}_2 \cdot \text{ReLU}\left(\mathbf{G}_1\mathbf{h}_u^{(K)} + \mathbf{g}\right)$
17 end
```

Figure 6: PinSage minibatch

The random walk algorithm is shown in Figure 7



Algorithm 1 Basic Random Walk; q is the query pin; E denotes the edges of graph G; α determines the length of walks; N is the total number of steps of the walk; V stores pin visit counts.

```
BasicRandomWalk($q$: Query pin, $E$: Set of edges, $\alpha$: Real, $N$: Int)
totSteps = 0, $V = \vec{0}$
repeat
    currPin = $q$
    currSteps = SampleWalkLength($\alpha$)
    for i = [1 : currSteps] do
        currBoard = $E$(currPin)[rand()]
        currPin = $E$(currBoard)[randNeighbor()]
        $V$[currPin]++
    totSteps += currSteps
until totSteps $\geq N$
return $V$
```

Figure 7: random walk

B Dataset

The movie rating files contain over 100 million ratings from 480 thousand randomly-chosen, anonymous Netflix customers over 17 thousand movie titles. The data were collected between October, 1998 and December, 2005 and reflect the distribution of all ratings received during this period. We use data from Netflix, there are total 17770 movies and CustomerIDs range from 1 to 2649429, with gaps.

C Structure

The pipeline structure is shown in Figure 8.

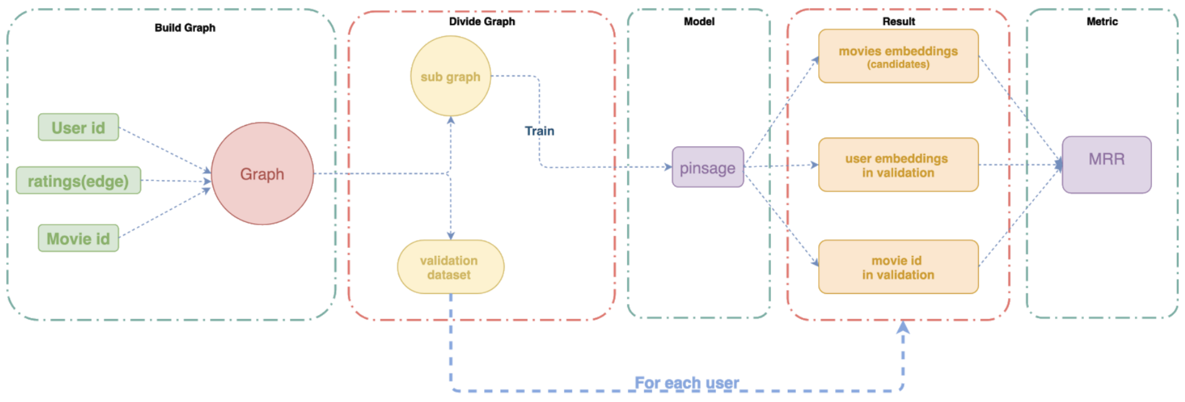


Figure 8: pipeline

We use the pinsgae code from dgl repository.
Suppose that a user ratings a movie, we can assume that there is an edge between this user and movie.
First, we use the users id and movie id as node and rating dataset as edges to build the whole graph. Then divide the dataset into train dataset, and test dataset. And use train dataset build the subgraph from the whole graph. Next, Put the sub graph into pinSage model to train, and output are embeddings for all users and movies.
For every user in validation: we get the candidate movies which means all the movies exclude those items which have edges with this user in train sub graph. To evaluate the similarity between this user and all movie candidates, we just calculate the Cosine similarity between this user's embedding and candidate movies' embedding.
Finally, we use the MRR as the metric.

D MRR

The mean reciprocal rank is a statistic measure for evaluating any process that produces a list of possible responses to a sample of queries, ordered by probability of correctness. The reciprocal rank of a query response is the multiplicative inverse of the rank of the first correct answer: 1 for first place,1/2 for second place, 1/3 for third place and so on. The mean reciprocal rank is the average of the reciprocal ranks of results for a sample of queries Q[4]:

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\text{rank}_i}$$

E Result

- By changing the negative sampling, we can get different performance, which is shown in 9.

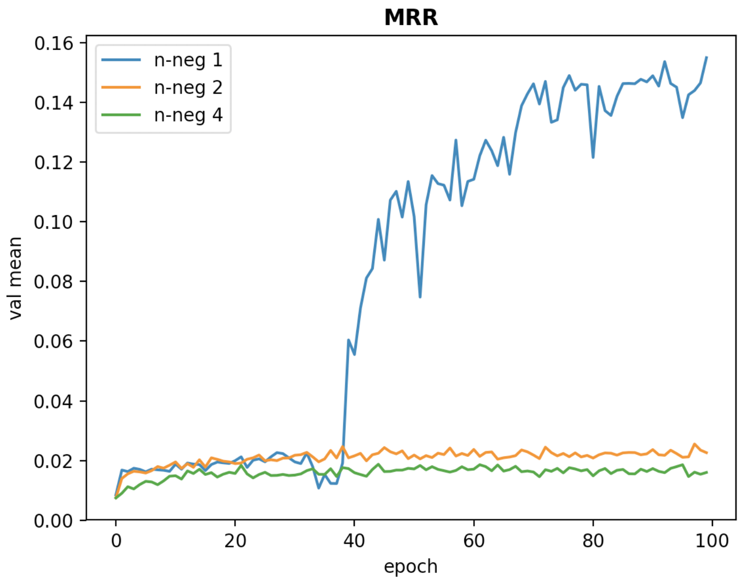


Figure 9: different negative sampling

Analysis: If we increase the size of negative sampling, the performance will become very poor in the same epoch.

- By changing the embedding size, we can get different performance, which is shown in 10.

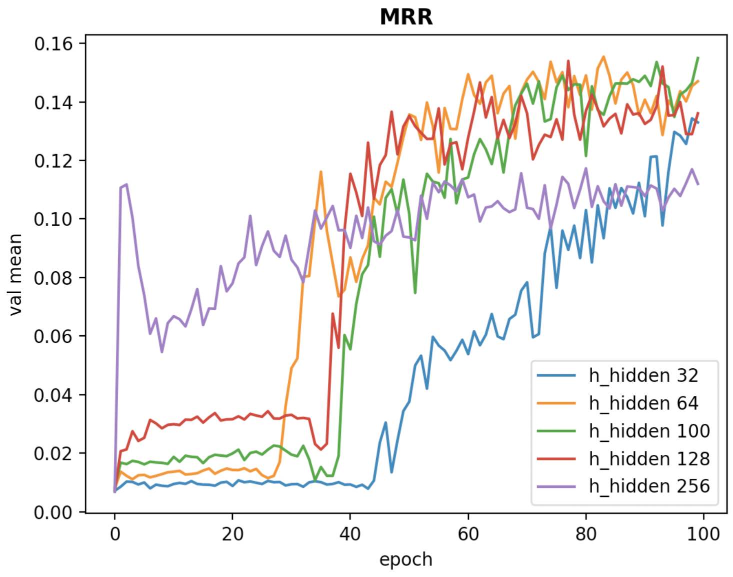


Figure 10: different embedding size

Analysis: The change of embedding size indeed has some influence in the first few epoch, but, with the number of epoch increase, this effect become smaller and smaller.But if we increase more,for instance,256 the performance will become much poor.

- By changing number of neighbors, we can get different performance, which is shown in 11.

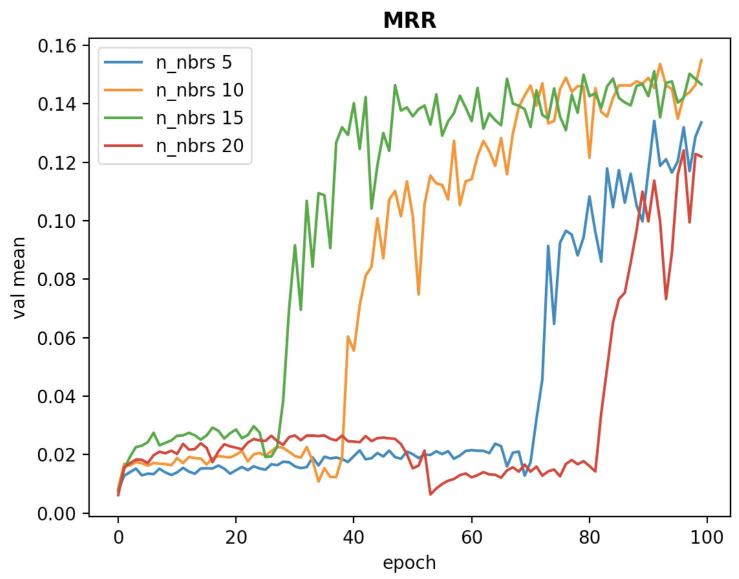


Figure 11: different neighbors

Analysis: Too many or too few neighbors can increase the training time.Because the model with different neighbors will get a similar performance in the end, but at different epoch. The reason is that: if we choose a small number of neighbors, the neighbor nodes cannot represent the node itself. If we choose a large number of neighbors, the training will become very inefficient. We should choose a Moderate neighborsIn this model,15 neighbors is a good choice.

- By changing the number of layers, we can get different performance, which is shown in 12.

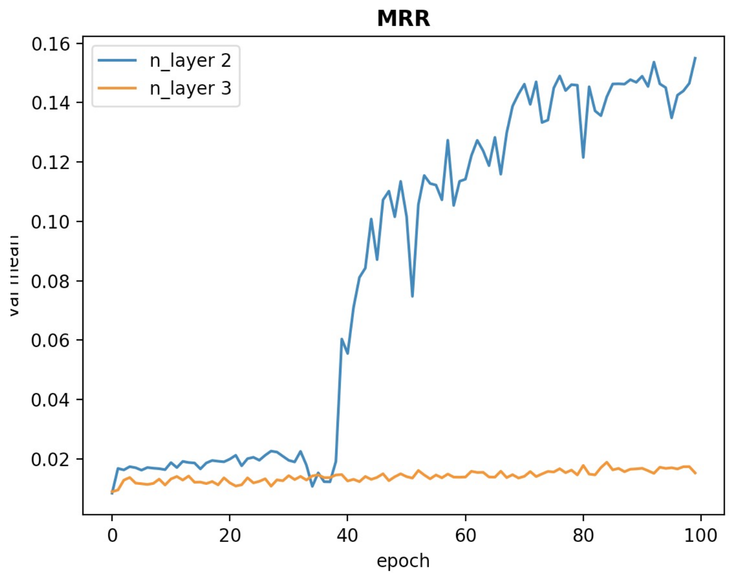


Figure 12: different layers

Analysis: if we increase the number of layers, the running time has increased a lot! From the picture, it is obviously that although we only increase one more layer, it will cause a great reduce on the performance. It may have a better performance with 3 layers if we train more epochs, but it will takes a lot of time which is inefficient.So the choice of a suitable layers is important.

References

[1] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*, 2017.

[2] Aleksandar Bojchevski and Stephan Gunnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*, 2018.

[3] Graph Convolutional Neural Networks for Web-Scale Recommender Systems *by Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, Jure Leskovec*

[4] MRR wiki