Final solution in assignment 2

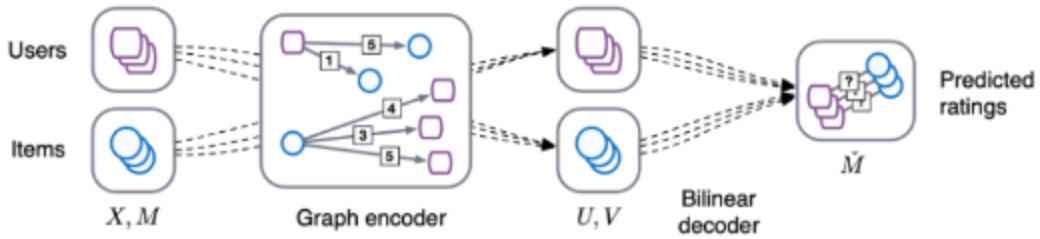
The formulated problem

Given the bipartite graph of users and movies. Between every user and movie there are 5 edges corresponding to ratings: 1,2,3,4,5. Every edge r has either value 1 (means user rated the movie as r) or 0 (means user rated the movie differently). Our task now is to predict, which edges have value 1 and which have value 0

The short overview of the solution

To solve the problem above, we use the GCMC (Graph convolutional matrix completion) with the following steps:

1. We learn an embedding for movies and users using encoder
2. We use those embeddings in bilinear decoder to produce a confidence map for every rating r showing the probability that user will give the movie rating r.
3. Using confidence maps, make a decision about rating from user to movie.



The model overview

Model advantages, disadvantages

According to “Papers with code”, this model is one of the best for the predictions in MovieLens100k, the difference in RMSE for this model and the best ever is 0.023 that is very small. This model utilizes everything: side features, graph structure. This model is easily scalable and changeable. What is more, this model is not too hard to understand and implement. Moreover, this model can be used on very large dataset with little changes stated in the paper. Finally, this model has no problem with cold start, when users have a few ratings, since once again we utilize the graph structure for predictions.

Therefore, model can find similar users to the given newbie and give him their well-rated movies.

As a result, it is hard to say specific disadvantages of the model. The original paper also does not state them. Since there are some implementation with little improvements in RMSE, the architecture is not awesome.

Initial embeddings

Authors proposed to use unique one-hot embeddings for users and items such that $\begin{pmatrix} X_u \\ X_v \end{pmatrix} = I$.

Encoder

Once again, we use encoder to learn an embedding. We learn them using convolution.

The following can be done inversely for messages from some user u to some item v.

Let's denote message from item v to user u through the edge of r as $m_{v->u,r} = W_r X_v$, where W_r is the linear transformation for rating r and X_v is the feature vector of item v.

Next, we can compute the received message by user u as the following:

$$m_{u,r} = \frac{1}{|N_{u,r}|} \sum m_{v->u,r} = \frac{1}{|N_{u,r}|} ((M_r)_{i,:} X_v W_r)^T$$

In words, we sum all the incoming messages and average them. The sum is expressed in matrix form, where $(M_r)_{i,:}$ is the adjacency matrix for rating r, contains only 1 or 0.

Then, for every end user, we stack the messages for every rating r and apply non-linearity on the result to get embedding for the concrete user:

$$h_u = \sigma(stack(\{m_{u,r}\}_{r \in \{1,2,3,4,5\}}))$$

Using both $h_u, m_{u,r}$ together, obtain that the new embeddings for users in matrix form are

$$H_u = \sigma(stack(\{D_u^{-1} M_r X_v W_r\}_{r \in \{1,2,3,4,5\}}))$$

Then, to add the influence of the nodes to their embedding, we add message from them to them as

$$H_u = stack(H_u, \{X_u W_r\}_{r \in \{1,2,3,4,5\}})$$

Next, in order to use side features, we add them in the similar manner. Denote X_u^f as side features of users. Then:

$$H_u = stack(H_u, \sigma(X_u^f W_u^f))$$

Finally, we obtain the embedding $H_u = \sigma(H_u W_u)$

Once again, the same logic is behind encoding the items embeddings.

Bilinear decoder

Since now we have embeddings for nodes, we can calculate the probabilities of having rating *r* between user *u* and item *v* (u,v being the corresponding embeddings here). The probability P_r of having rating *r* is

$$P_{r,u,v} = \frac{e^{u^T Q_r v}}{\sum_{s \in \{1,2,3,4,5\}} e^{u^T Q_s v}}$$

where Qs are the trainable parameters. Having a confidence maps, we can calculate the rating predictions simply as $M = \sum_{r \in R} r P_r$, or the expectation. P_r here is the probability map for rating r.

Loss and metric

Authors of the original paper proposed to use Cross-entropy loss for the problem, since as the output of the model we have the probability distribution over classes(ratings) for every pair of user and item. Since we also have a matrix-completion problem, I will track this metric too. However, RMSE is not used for training.

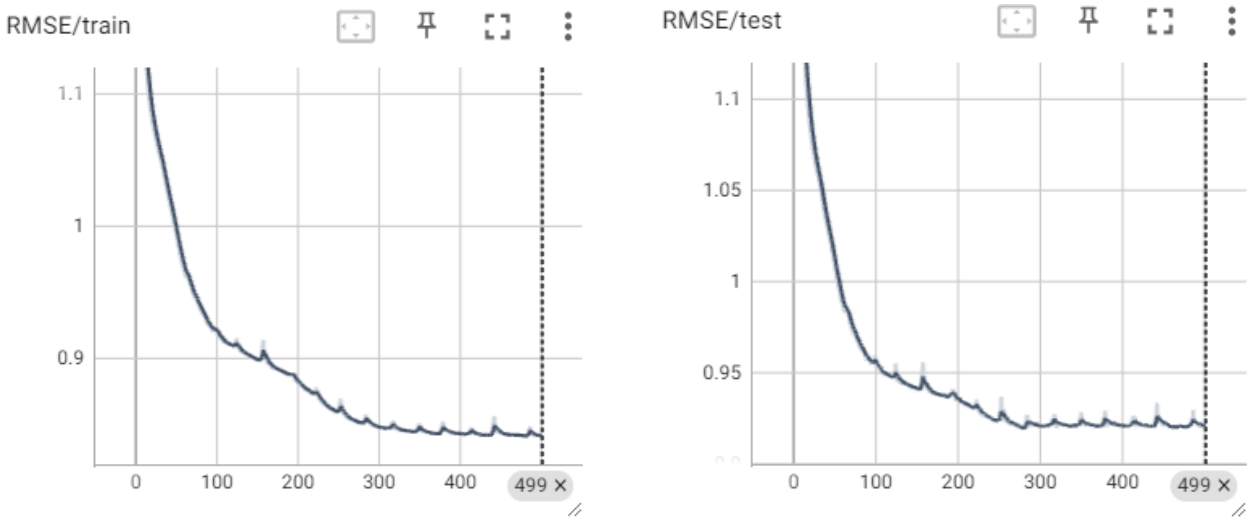
Training

Following the paper, I used full-batch training with Adam with learning rate of 0.01. Also, weight decay was set to 0.005. The number of epochs is 500. The hidden dimension

everywhere is 5. The training was done on the full dataset (in form of the rating matrix) with 80/20 split/masking (20% of random cells with the present rating, absent ratings are ignored).

Results

Here you can see the RMSE during the training from tensorboard:



We can see that on ~270 epoch, model stopped to learn.

The best RMSE on test data is 0.9187.

Benchmark

I computed MAP@K on the test split from the training. There are the results for different k:

K=5	K=10	K=20
0.7706	0.7729	0.7821

References

1. Rianne van den Berg and Thomas N. Kipf and Max Welling. Graph convolutional matrix completion. 2017
2. Mingbo Cui et.al. Graph-based recommender system, 2019