CSC413 Final Project

Sanzhe Feng, Hehan Zhao
Department of Computer Science
University of Toronto

Abstract

The paper reconstruct two different recommendation system models and extend them in different tasks/benchmarks. The code could be found on Github [1]

1 Introduction

Recommendation systems are a type of artificial intelligence (AI) that provide personalized suggestions to users based on their preferences, behavior, and other factors[1]. These systems have become an essential component of the digital landscape, driving user engagement and satisfaction on different platforms. They can be used in a wide range of applications, from suggesting products or content to recommending connections on social media apps.

Recommendation systems are unique and irreplaceable. By providing personalized recommendations, users can easily discover relevant content or products, which leads to increased satisfaction and engagement and saving them a lot of time and effort. For businesses, recommendation systems can help boost sales by promoting products or services that are likely to appeal to individual customers, potentially increase revenues.

Deep learning is a particularly suitable choice for recommendation systems because of its ability to model complex patterns and relationships in data. By leveraging multi-layered neural networks, deep learning can automatically learn to capture the underlying structure of user preferences and item features, often leading to more accurate and reliable recommendations[2].

In this project, we will reconstruct two of the best methods for the MovieLens 1M benchmark (See Appendix for details)[3]: 1. *GLocal-K: Global and Local Kernels for Recommender Systems* by Han et al[4], and 2. *Kernelized Synaptic Weight Matrices* by Muller et al, presented at the International Conference on Machine Learning (ICML) in 2018 [5]. Besides, we will also explore possible extensions on these methods, including their applications on a new benchmark. The metric that used to evaluate the models is the Root Mean Squared Error (RMSE) (See Appendix for details).

1.1 GLocal-K: global and local kernels for recommender systems

The authors presents a novel kernel-based approach for building recommender systems. It focuses on combining the strengths of both global and local kernels through a series of algorithms to improve the model's performance in predicting user preferences for items.

The GLocal-K framework employs global kernels to capture the overall structure and relationships between users and items by modeling high-level interactions. On the other hand, local kernels are used to concentrate on specific regions or neighborhoods in the feature space, allowing for a more fine-grained understanding of user-item interactions. The algorithms used in the paper seamlessly integrate these two types of kernels, resulting in a more robust model capable of learning complex patterns and generalizing well to unseen data. By leveraging these algorithms, GLocal-K has the

[1]`https://github.com/Perseids3/CSC413_Final_Project.git`

potential to surpass traditional recommender systems that rely solely on either global or local kernels, ultimately leading to more accurate and personalized recommendations for users.

1.2 Kernelized synaptic weight matrices

The authors present a method that leverages the kernel trick to learn synaptic weight matrices in a more expressive manner, which leads to improved performance in various tasks, including collaborative filtering.

A key contribution of the paper is the introduction of Sparse Fully Connected (Sparse FC) layers. These layers are designed to work with sparse input data and exploit the sparsity to reduce computational complexity. In the context of collaborative filtering, this is particularly useful, as user-item rating matrices are often very sparse.

The paper also introduces KernelNet, a neural network that incorporates Sparse FC layers and kernelized synaptic weight matrices. KernelNet is designed to learn an implicit feature space in which users and items are represented, leading to more accurate recommendations. By utilizing the kernel trick, KernelNet can learn non-linear relationships between users and items, increasing its expressiveness and modeling capability.

2 GLocal-K

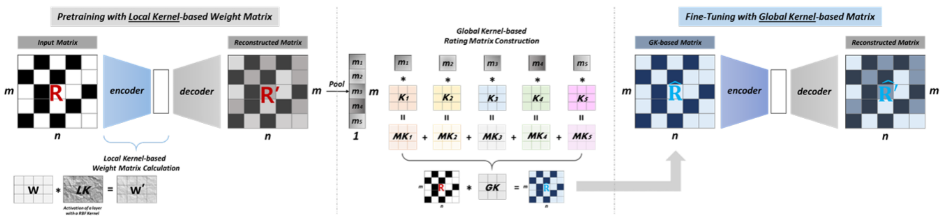
Pretraining with Local Kernel-based Weight Matrix
Input Matrix
R
m
n
encoder
decoder
Reconstructed Matrix
R′
Local Kernel-based Weight Matrix Calculation
W
LK
W′
Activation of a layer with a RBF Kernel
Pool
m1
m2
m3
m4
m5
1
Global Kernel-based Rating Matrix Construction
K1
K2
K3
K4
K5
MK1
MK2
MK3
MK4
MK5
GK
Fine-Tuning with Global Kernel-based Matrix
GK-based Matrix
R̂
R̂′

Figure 1: G-LocalK architecture from the paper

As demonstrated in Fig 1, this architecture is made up with two components: Pre-training with local Kernel and fine-tuning with global kernel[4].

For pre-training, we first train an item-based autoencoder (AE), to reconstructed r'_i to predict missing ratings given r_i. The model can be represented as:

$$r'_i = f(W^{(d)} \cdot g(W^{(e)} r_i + b) + b'),$$

where $W^{(e)} \in \mathbb{R}^{h \times m}$ and $W^{(d)} \in \mathbb{R}^{m \times h}$ are weight matrices, $b \in \mathbb{R}^h$ and $b' \in \mathbb{R}^m$ are bias vectors, and $f(\cdot)$ and $g(\cdot)$ are non-linear activation functions.

The Autoencoder uses an auto-associative neural network with a hidden layer of size h. It reparametrizes the weight matrices using a radial basis function (RBF) kernel defined as:

$$K_{ij}(U, V) = \max(0, 1 - ||u_i - v_j||_2^2),$$

$K(\cdot)$ is computes the similarity between , u_i and v_j. Then, we compute a local kernelised weight matrix as:

$$W'_{ij} = W_{ij} \cdot K_{ij}(U, V)$$

where W' is computed by the Hadamard-product.

Next we define the global convolutional kernel to create the rating matrix:

$$\mu_i = \text{avgpool}(r'_i)$$

$$\text{GK} = \sum_{i=1}^{m} \mu_i \cdot k_i$$

$$\hat{R} = R \otimes \text{GK}$$

where the aggregated kernel GK $\in \mathbb{R}^{t \times t}$ is the global convolution kernel and $\hat{\text{R}}$ is the global kernel-based rating matrix.

We then start the fine-tuning process using global kernel-based rating matrix$\hat{\text{R}}$ as input. It will take weights of a pre-trained AE model and makes an adjustment of the model based on the global kernel-based rating matrix, as depicted in Figure 1. Detailed explanation of the architecture can be found at the original paper. Here we only include the key ideas.

2.1 Experiment and results

2.1.1 General performance

Based on our results in Table 1 (Appendix), we can see that there is a difference about 0.15 between our model (0.975) and the original work(0.822). We think it is because of: 1. the results given in the paper were the best results while we picked ours from only random trail. 2. We reconstructed the model using pytorch instead of tensorflow (presented in the original code), which restricts us from using the same optimizer.

2.1.2 Effect of global kernel

We also performed some testing on the size of kernel 1. Similar to the results from original paper(0.822), we saw that when we used a $3x3$ kernel, the RMSE will be very close to the results using global kernel(0.975).

2.2 Sensitivity analysis

We tuned some hyperparameters, including number of epoches, number of hidden layers, hidden layer dimension, different optimizers and learning rates, to test on the sensitivity of this architecture. The results were as expected: Only increase in epoch (two times) will lead to a arguable decrease in RMSE (around 0.08). Other hyperparameters all gave equivalent and even worse results (Table 3 in Appendix).

2.3 Extension on GLocal-K

As suggested in the paper, our architecture can be applied to other recommendation systems like MovieLens 100K and Douban. Here is the result for using MovieLens 100K benchmark (Table 2 in Appendix). While there is still a difference of 0.07, the testing results are as expected (original res: 0.89; our res 0.96).

2.3.1 GCN

In addition to our current architecture, there exist certain tasks that have demonstrated the improved prediction capability of global kernels. One such task is semantic segmentation. However, due to constraints in time and technical resources, we have not implemented a full-fledged Global Convolutional Network (GCN) as presented in the paper by Peng, et al. [6] for this project. Instead, we have developed a simplified example to showcase only the functionality of GCN on the CIFAR-10 dataset. Nonetheless, the success of GCN solidly proved the effectiveness of global kernels for specific tasks.

3 Sparse FC

The model is an item-based autoencoder. First a kernelNet-Layer is used with following kernel:

$$K_{\sigma}(\vec{u}, \vec{v}) = tanh(\vec{u}, \vec{v})$$

Remember that the finite support kernel:

$$K_{fs}(\vec{u}_i, \vec{v}_j) = max(0, 1 - a \cdot D(\vec{u}_i, \vec{v}_j))$$

Now with finite-support kernel and the Hadamard-product of a dense connection matrix with a kernelized weight matrix,the full model is :

$$h(\vec{r}, \theta) = f(\mathbf{W} \cdot g(\mathbf{V}\vec{r} + \vec{\mu}) + \vec{b})$$

where the weight matrices either take the form:

$$V_{ij} = \alpha_i K_\sigma(\vec{u}_i, \vec{v}_j)$$

$$W_{ij} = \beta_i K_\sigma(\vec{s}_i, \vec{t}_j)$$

or in the form using K_{fs}, which refers to as "sparse fully-connected":

$$V_{ij} = \alpha_{ij} K_{fs}(\vec{v}_i, \vec{u}_j)$$

$$W_{ij} = \beta_{ij} K_{fs}(\vec{s}_i, \vec{t}_j)$$

To minimized a regularized square-error, L-BFGS-B and RPROP optimizers are used, the regularization term added to cost is:

$$R = \lambda_2(W_{ij}^2 + V_{ij}^2)$$

while in the sparse fully-connected case this is:

$$R = \lambda_2(\sum_{ij} \alpha_{ij} + \sum_{ij} \beta_{ij}) + \lambda_0(\sum_{ij} K(\vec{u}_i, \vec{v}_j) + \sum_{ij} K(\vec{s}_i, \vec{t}_j))$$

Wile λ_2 can be treated as L_2 regularization and λ_0 as sparsity "L_0" regularization parameters.

3.1 Experiment and results

In the model reconstruted, the best RMSE got is 1.037, while in the original paper, the bet RMSE against ML-1M is 0.824.

As the original paper splits the training and validation sets randomly, and the reconstruction of the original paper used PyTorch in replace of Tensorflow 1.13, so it is acceptable that the experimatal results of reconstruction underperforms the paper model. More details can be found in Appendix: Sparse FC experimental results section.

3.2 Sensitivity analysis

The model's performance is sensitive to the number of hidden units, but with diminishing returns as the number of units increases, while a smaller learning rate yields better results. However, adding the number of layers seem to hurt the performance a little bit (Table 4 in Appendix).

3.3 Extension on Sparse FC

The Sparse FC is evaluated against Jester benchmark[7] data_1 with the RMSE result 9.75. However, as the joke ratings range from -10 to 10 in the dataset. A RMSE of 9.75 indicates that the predicted ratings are, on average, 9.75 points away from the true ratings, which is quite large given the scale of the ratings. The bad performace of Sparse FC model might due to that the Jester dataset is highly sparse, with users rating only a small subset of jokes. Also, Sparse FC networks do not perform automatic feature learning like deep learning-based approaches. As a result, Sparse FC networks may not be able to effectively capture the underlying patterns in such sparse data as they focus on reducing the complexity of the model rather than on modeling complex patterns or relationships.

4 Conclusion

Deep learning-based recommendation systems present a potent approach for delivering personalized and precise recommendations. In this paper, we analyzed and reconstructed two such models, demonstrating their effectiveness on the MovieLens 1M benchmark. Nevertheless, while these models performed admirably on the MovieLens dataset, their performance metrics were less impressive when applied to different tasks and benchmarks. This highlights the importance of further research and development to enhance the adaptability and generalizability of deep learning-based recommendation models across a broader range of applications and datasets.

References

[1] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender Systems Handbook*, pages 1–35. Springer, 2011.

[2] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.

[3] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. In *ACM Transactions on Interactive Intelligent Systems (TiiS)*, volume 5, page 19. ACM, 2016.

[4] Long Han, Lim, Burgstaller, and Poon. Glocal-k: Global and local kernels for recommender systems. 2021.

[5] Lorenz Muller, Julien Martel, and Giacomo Indiveri. Kernelized synaptic weight matrices. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, pages 3637–3646, Stockholmsmässan, Stockholm, Sweden, July 10-15 2018.

[6] Yu Luo Peng, Zhang and Sun. Large kernel matters – improve semantic segmentation by global convolutional network. 2017.

[7] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[8] Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate Research*, 30(1):79–82, 2005.

[9] Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

Table 1: Performance of our reconstructed model in RMSE

	Our Results	Paper's Results
Overall	0.975	0.822
Fixed-size Kernel(3x3)	0.975	0.822

Table 2: Performance of our reconstructed model in RMSE using MovieLens 100K benchmark

	Our Results	Paper's Results
Overall	0.960	0.890
Fixed-size Kernel(3x3)	0.959	0.890

Appendix

Benchmarks

The MovieLens 1M dataset is a widely-used benchmark for collaborative filtering and recommendation systems. It contains 1 million ratings from 6,000 users on 4,000 movies[3]. Similarly to MovieLens, Jester dataset is a benchmark used for evaluating collaborative filtering algorithms, specifically designed for joke recommendation systems. The dataset was introduced by Ken Goldberg and his colleagues at the University of California, Berkeley in 2001. It contains anonymous joke ratings from users, where each user rates a subset of jokes on a continuous scale from -10 (least funny) to 10 (most funny). The primary goal of recommendation systems built using the Jester dataset is to predict users' joke preferences and recommend jokes they are likely to find funny[7].

Root Mean Square Error(RMSE)

The Root Mean Squared Error (RMSE) is a popular metric used to evaluate the performance of recommendation systems. It measures the differences between the predicted values and the actual values, providing an aggregated measure of the model's accuracy. The lower the RMSE, the better the model's performance in terms of prediction accuracy[8].

To compute the RMSE, first calculate the squared differences between the predicted values and the actual values. Then, take the mean of these squared differences, and finally compute the square root of the mean. Mathematically, it can be represented as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of data points[9].

Sparse FC experimental results

The model is trained to prefict movie ratings of the MovieLens 1M, the datasets are highly sparse (density 0.013/0.045/0.059). The code randomly designate 10% of the given ratings asvalidation data. The hyperparameters of Sparse FC model are:

1. n_hid: The number of hidden units in each layer of the network. It is set to 500 in the original paper.
2. lambda_2: The regularization strength for the sparsifying kernel. It is set to 0.013 in the original paper.
3. lambda_m: The L2 regularization strength. It is set to 60 in the original paper.

Table 3: Sensitivity Analysis For GLocal-K using 100K dataset

Default(RMSE)	n_epoch	n_layers	n_hid	Optimizer	lr
1.03	x1.5: 0.97	x2: 1.06	x2: 1.03	SGD: NA	x10: 1.03
	x2: 0.95			RMSprop: 1.06	x50: 1.04

Table 4: Hyperparameters set for Sparse FC model

	n_hid	lambda_2	lambda_m	n_layers	output_every	n_epoch	lr	RMSE
Model_1	500	0.013	60	2	50	1000	0.1	1.059
Model_2	600	0.013	60	2	50	1000	0.1	1.044
Model_3	700	0.013	60	2	50	1000	0.01	1.037
Model_4	700	0.013	60	3	50	1500	0.01	1.090

4. n_layers: The number of layers in the network. It is set to 2 in the original paper.
5. output_every: The frequency (in terms of the number of epochs) at which the performance on the test set is evaluated. It is set to 50 in the original paper.
6. n_epoch: The total number of epochs to train the network. It is set to n_layers * 10 * output_every in the original paper, which evaluates to 1000.
7. lr: The learning rate for the L-BFGS optimizer. In the original paper it is set to 0.1.

Some typical results as well as the hyperparaters value is listed in Table 4

Contribution

Hehan Zhao: Sparse FC
Sanzhe Feng: GLocal-K