Recommender System using Neural Collaborative Filtering

Kai Fang z5137591
Yihao Wu z5154763
Dan Liu z5094271
Hao Huang z5112059

1. INTRODUCTION

In recent year, collaborative filtering(CF) has been successfully used to provide users with personalized products and services. Among the various collaborative filtering technologies, matrix factorization(MF) is the most popular one. However, the performance of the interaction between user and item features is hindered by the simple choice of the inner product of their latent vectors. The inner product, which simply combine the multiplication of latent features linearly, may not be sufficient to capture the complex structure of user interaction data.
To tackle such issue, we strive to develop technologies based on neural networks which has been proved to be capable of approximating any continuous function to tackle the key problem in CF. The main contribution is to combine SVD and MLP model as our model by utilizing *DNNs* to extra non-linear and implicit user-item interaction.

2. METHOD

2.1 Matrix Factorization

2.1.1 Basic Matrix Factorization Model

Basic-CF algorithms offer a approach to use the similarity between items and users to make predictions based on user-item matrix. Although Basic-CF algorithms are intuitive, it hard to implement and inefficient to handle large, sparse data.
Matrix Factorization techniques not only process dimensionlity reduction but also discover two latent features which could be understand as hidden features under the user and item. The main idea of MF is to map both users and items to a joint latent factor space of dimensionality k, such that it constructs P_u as user-latent matrix and Q_i as item-latent matrix, and let its interactions $P_u^T Q_i$ as $\hat{R}$ fit the original matrix R as much as possible. So that we could predict unknown value in original matrix by taking inner product of $u \cdot i | u \in P_u, i \in Q_i$ learned from observed values.
For movies case ,we can represent predicted rates $\hat{r_{ij}}$ as:

$$\hat{r_{ui}} = P_u^T Q_i = \sum_{k=1}^{k} P_{uk} Q_{ki} \quad (1)$$

we assume k represent the number of latent features and k would be less than either u or i.

2.1.2 Matrix Factorization Model with Biases

However, Equation (1) might unable to simulate the psychical truth of data cause observed values usaully have subjective characters which are independent to others either users or items. known as bias.
Thus, we extend our methods by adding bias b_{ui} as follows:

$$b_{ui} = b_u + b_i + \mu$$

Our final $\hat{r_{ui}}$ can be represented as follows:

$$\hat{r_{ui}} = P_u^T Q_i + b_u + b_i + \mu \quad (2)$$

The parameters b_u and b_i indicate the observed deviations of item and user separately from the average of whole rates μ.
Finally, with the regularization, to learn the factor vectors (P_u, Q_i, b_u, b_i),we define our objective function as form of minimized regularized squared error:

$$\min_{P_u, Q_i, b_u, b_i} \sum_{(u,i) \in Z} (r_{ui} - P_u^T Q_i - b_u - b_i - \mu) + \lambda(\|p_u\|^2 + \|Q_i\|^2 + b_u^2 + b_i^2) \quad (3)$$

$$z = (u, i) : r_{u,i} \neq 0$$

As we mentioned, λ is a constant parameter to avoid overfitting by control the magnitudes of regularization and usually determined by cross-validation, it can generate a good approximation even if model get smaller numbers in $\hat{r_{ui}} - r_{ui}$.
In our implementation, λ is set as 0.15.

2.1.3 Learning Algotithms

To minimize Equation (3), we using Stochastic Gradient Descent (SGD) to loop through all ratings in the train data and compute the associated prediction error

$$e_{ui} = r_{ui} - P_u^T Q_i$$

Then, we obtain four update rules for P_u, Q_i, b_u, b_i, which are:

$$b_u \leftarrow b_u + \gamma(e_{ui} - \lambda b_u)$$
$$b_i \leftarrow b_i + \gamma(e_{ui} - \lambda b_i)$$
$$P_u \leftarrow p_u + \gamma(e_{ui} \cdot Q_i - \lambda P_u)$$
$$Q_i \leftarrow Q_i + \gamma(e_{ui} \cdot P_u - \lambda Q_i)$$

γ is define as learning rate in gradient descent, in our implementation, we set as 0.04.

2.1.4 Limitation of Matrix Factorization

Obviously, MF linearly combines item-latent matrix and user-latent matrix with same weight. it can be deemed as a linear model of latent factors. it might not be appropriate for the real situation cause two low-dimensional latent space could be too weekness to estimate the complex user-item interactions. However, MF could not resolve this issue by increasing the amount of latent factors k cause MF is very slow for a large, dense user-item matrix.
Thus, in our work, we deal with this limitations by learning user-item interactions using deep neural networks.

2.2 Multilayer perceptron

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. An MLP consists of at least three layers of nodes. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

2.2.1 Layers

The MLP consists of three or more layers (an input and an output layer with one or more hidden layers) of nonlinearly-activating nodes making it a deep neural network. Since MLPs are fully connected, each node in one layer connects with a certain weight w_{ij} and a bias b_i to every node in the following layer.
As for the design of network structure, a common solution is to follow a tower pattern, where the bottom layer is the widest and each successive layer has a smaller number of neurons (as in Figure 1). The premise is that by using a small number of hidden units for higher layers, they can learn more abstractive features of data. We empirically implement the tower structure, halving the layer size for each successive higher layer.

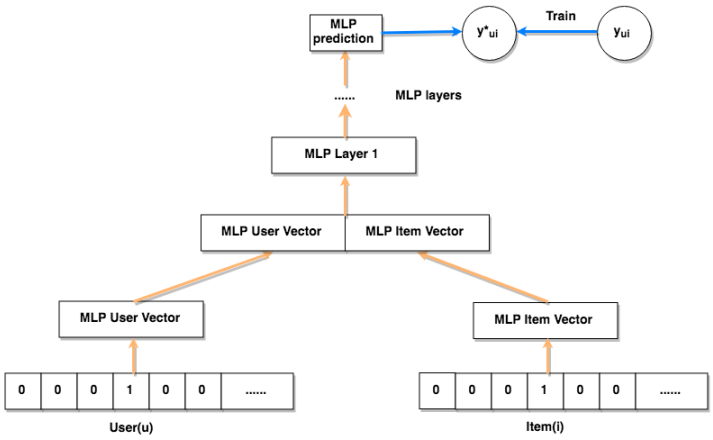

Figure 1

For this project, the input layer is the concatenation of latent vectors of user and item and transform it to a binarized spares vector with one-hot encoding.

$$l_1 = \phi_1(p_u, q_i) = \begin{bmatrix} p_u \\ q_i \end{bmatrix}$$

And we choice to use two hidden layers

$$l_2 = \phi_2(l_1) = a_1(W_1^T l_1 + b_1)$$
$$l_3 = \phi_3(l_2) = a_2(W_2^T l_2 + b_2)$$

The output layer is

$$l_4 = \phi_4(l_3) = a_3(W_3^T l_3 + b_3)$$

2.2.2 Activation Function

For activation function a_i, we can choose sigmoid or hyperbolic tangent (tanh). The sigmoid function restricts each neuron to be in (0,1), which may limit the model's performance; and it is known to suffer from saturation, where neurons stop learning when their output is near either 0 or 1. So we choose tanh function because it typically yields to faster training (and sometimes also to better local minima).

$$a(x) = \frac{(e^x - e^{-x})}{e^x + e^{-x}}$$

2.2.3 Loss function

In this project, we choose mean square error (mse) as the loss function. For each layer, we use mini-batch gradient descent to update weight w_{ij} to converge mse. And we choose Adam as the optimization of gradient descent.

2.3 Fusion of SVD and MLP

So far, we have developed two training models to capture the similarity between users and items. One model is SVD, which applies a linear kernel to model the latent feature interactions, and another one is MLP, which uses a non-linear kernel to learn the interaction function from data. One idea then arises: Whether we can ensemble SVD and MLP to better model the complex user-item interactions by mutually reinforce each other.
To provide more flexibility to the ensemble model, we design SVD and MLP to learn separate embeddings as our bottom input instead of to share the same embedding layer. This point may less likely to limit the performance of the fused model. For example, the fused model implies that SVD and MLP must use the same size of embeddings; for training data where the optimal embedding size of the two models may different, this solution may fail to obtain the optimal ensemble model. Then, combining the two models by concatenating their last hidden layer as the input of the final output layer, which is predicted score $\hat{y}_{ui}$ and training is performed by minimizing the mean squared error (MSE) between $\hat{y}_{ui}$ and its target value y_{ui} and plus the regularization terms in **Mini-batch Gradient Descent** way.

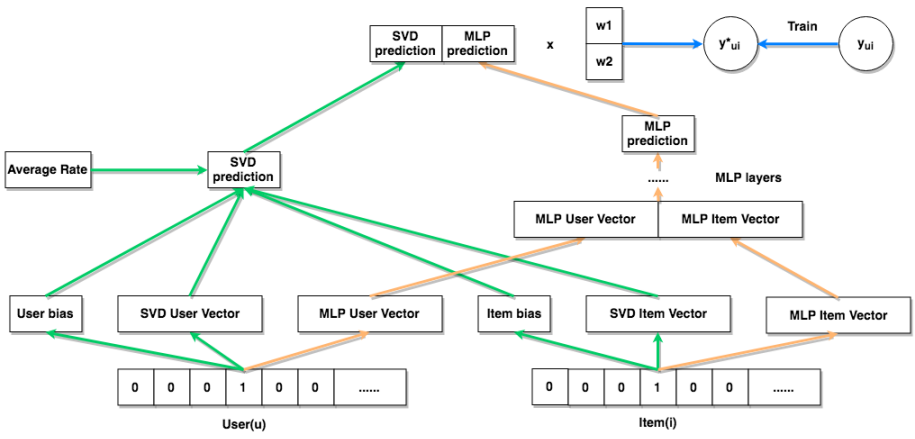

Figure 2

Figure 2 illustrates our main point, the formulation of which is given as follows:

$$\phi^{SVD} = p_u^{SVD} \odot q_i^{SVD}$$

$$\phi^{MLP} = a_L(W_L^T(a_{L-1}(\ldots a_2(W_2^T \begin{bmatrix} p_u^{MLP} \\ q_i^{MLP} \end{bmatrix} + b_2)\ldots)) + b_L)$$

$$\hat{y_{ui}} = \sigma(h^T \begin{bmatrix} p_u^{MLP} \\ q_i^{MLP} \end{bmatrix})$$

$$\min_{p_u^{SVD}, Q_i^{SVD}, p_u^{MLP}, Q_i^{MLP}, b_u, b_i} MSE(y_{ui}, \hat{y_{ui}}) + \lambda_1(\|P_u^{SVD}\|^2 + \|Q_i^{SVD}\|^2) + \lambda_2(\|P_u^{MLP}\|^2 + \|Q_i^{MLP}\|^2) + \lambda_3(|b_u| + |b_i|)$$

In which notations of q_i^{SVD} and q_i^{MLP} acted as item embedding for SVD and MLP parts, respectively; and P_u^{SVD} and P_u^{MLP} denote the user embeddings. As discussed before, we use **hyperbolic tangent (tanh)** as the activation function of MLP layers. The ensemble model thus combines the linearity of SVD and non-linearity of MLP for modelling user-item latent structures. Each model parameter can be calculated with standard back-propagation and using the same way as MLP model to coverage loss.

3. EXPERIMENTS

In this section, we test our model with two real-world datasets.

Table 1: Datasets Statistics

dataset	#moives	#user	#ratings	density (%)
MovieLens 100k	1682	943	100000	6.30
MovieLens 1M	3706	6040	1000209	4.46

3.1 Experimental Setup

3.1.1 Dataset Description

3.1.2 Evaluation Matrics

we refer to **Root Mean Squared Error (RMSE)** as our evaluation metric to measure the model accuracy. It is defined as:

$$RMSE = \sqrt{\frac{1}{|T|}\sum_{(u,i)\in T}(\hat{r_{ui}} - r_{ui})^2}$$

$|T|$ is the whole number of ratings in the test dataset, r_{ui} is the ground truth, $\hat{r_{ui}}$ denotes the predicted ratings for T.

3.2 Experimental Results

we select baseline methods including user-based CF, item-based CF, NMF, PMF, BiasedSVD and SVD++, six

relevant common models as experiment comparision. For train and test data selection, we attempt two ways: 90% or 50 % allocate for training and remaining allocate for testing.

Table 2: Average RMSE for Movielens-100k and Movielens- 1M

Method	**ML-100K**	**Method**	**ML-1M**
	90% 50%		90% 50%
User-based CF	1.224 1.242	User-based CF	1.115 1.327
Item-based CF	0.921 0.958	Item-based CF	0.885 0.900
NMF	0.958 0.997	NMF	0.915 0.927
PMF	0.952 0.977	PMF	0.883 0.890
Biased SVD	0.911 0.936	Biased SVD	0.876 0.889
SVD++	0.913 0.938	SVD++	0.855 0.884
MLP	0.895 0.928	MLP	0.862 0.880
SVD_MLP	0.889 0.922	SVD_MLP	0.855 0.875

Based on the evaluation results in Table 2, we found that our model MLP and SVD_MLP achieve good performance than above six common methods. Besides, SVD_MLP model perfoms slightly better than MLP model when the dataset get larger.

4. CONCLUSION

In this report, we present two efficient CF models, namely MLP and the fusion of SVD and MLP which are explore neural network architectures for collaborative filtering. They are able to learn complex and implicit interaction between users and items in different ways. In our experimentation part, we can see our models achieves the mostly best performance on the datasets compared with basic CF, SVD and SVD++.
However, we can further optimize our model in following aspects:

1. Pre-training
   We first train SVD and MLP with random initializations until convergence, and then use their model parameters as the initialization for the corresponding part of the ensemble model. The reason why its works is that gradient-based optimization methods only find the locally-optimal solution.
2. Using content features to represent users and items
   The bottom input layers can be customized to support a wide range of modelling of users and items, such as context-based, content-based and neighbor-based. Such a generic feature representation for inputs can be easily adjusted to address the cold-start problem by using content features to represent users and items.

5. REFERENCES

1. Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. IEEE Computer, 42(8):30–37, 2009.
2. S. Rendle. Factorization machines. In ICDM, pages 995–1000, 2010.
3. R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In NIPS, pages 1–8, 2008.
4. R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In ICDM, pages 791–798, 2007.
5. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In WWW, pages 285–295, 2001.

6. R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In NIPS, pages 926–934, 2013.
7. Xiangnan He , Lizi Liao , Hanwang Zhang , Liqiang Nie , Xia Hu , Tat-Seng Chua, Neural Collaborative Filtering, Proceedings of the 26th International Conference on World Wide Web, April 03-07, 2017.
8. H. Wang, N. Wang, and D.-Y. Yeung. Collaborative deep learning for recommender systems. In KDD, pages 1235–1244, 2015.
9. M. Wang, X. Liu, and X. Wu. Visual classification by l1 hypergraph modeling. IEEE Transactions on Knowledge and Data Engineering, 27(9):2564–2574, 2015.
10. Y. Koren. Factorization meets the neighborhood a multifaceted collaborative filtering model. In KDD 2008, Las Vegas, USA.