Project report

Team name: M2 Robo

Lee Chun Yin
3035469140

Chiu Yu Ying
3035477630

Chan Kwan Yin
3035466978
Team leader

1. Introduction

1.1. collaborative filtering (CF)

CF is a technique for recommendation system, in which historical feedback data are used to infer connections between users and products [3]. While additional features can be introduced to offset certain bias effects [1], two inputs (user and product) and one output (user rating on the product) are generally sufficient to train a CF model without involving domain-specific data.

Two major approaches for CF include neighbourhood models and latent factor models. Neighbourhood models compare the similarity between users and recommend products positively rated by similar users, while latent factor models perform dimensional reduction on both users and movies to a common, smaller set of feature attributes such that users are recommended with movies of more coherent features.

1.2. The Netflix Prize dataset

The Netflix Prize is a competition for the prediction of users' favour of movies. The dataset provides existing ratings of users on given movies, and models are trained to predict new ratings.

1.2.1 Dataset format

The dataset contains 100480507 rows of data structured in the following format:

User ID	490189 discrete values
Movie ID	17770 discrete values
Rating	1, 2, 3, 4, 5
Date	Dates from 1999 to 2005

1.2.2 Distribution of ratings

Ratings are mostly distributed around 3 and 4, as shown in Figure 1.

Except for some extreme cases, the number of ratings per user over the 7 years mostly follow an exponential relation for users from 10 to 1000 ratings, as shown in Figure 2. The top 10 users with the highest number of ratings range from 17651 to 8877.

For movies with at least 100 ratings (which is the case for the majority), their numbers of ratings demonstrate a similar but more concave relationship, as shown in Figure 3.

1.2.3 Evaluation process

To evaluate performance, 1425333 rows (about 1.42% of all data) are speified as the standard "probe". We perform analysis in the following procedure:
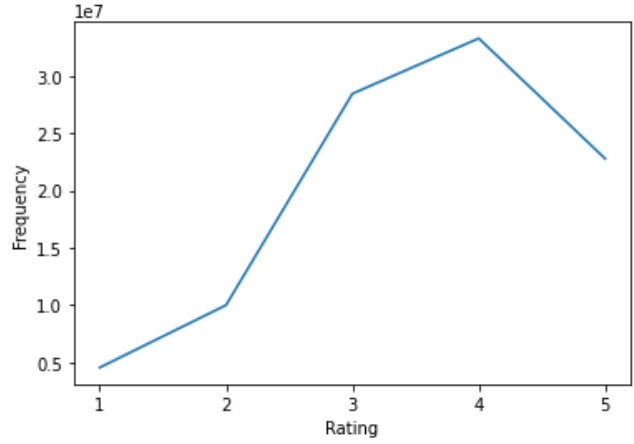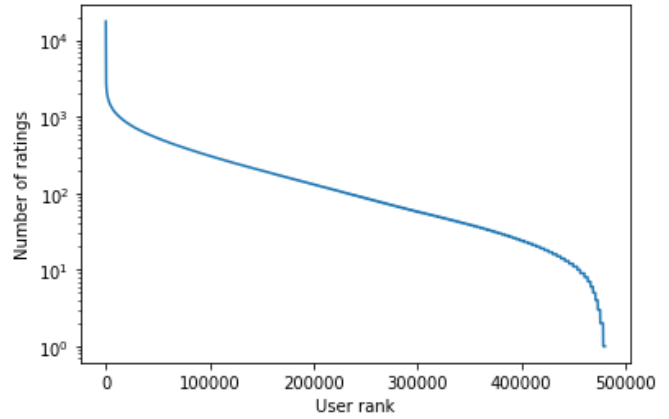

Figure 1. Frequency of ratings
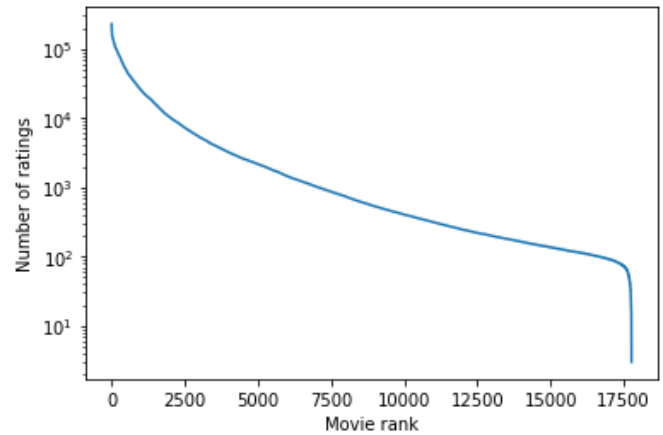

Figure 2. Number of ratings per user

Figure 3. Number of ratings per movie

1. Train the model with the 99055174 non-probe rows.

2. Predict user ratings with the 1425333 probe rows.

3. Compute the root mean squared error (RMSE) be-

tween the predicted data and actual data.

$$\text{RMSE} = \sqrt{\sum_{(i,j) \in E} \frac{(\hat{r}_{ij} - r_{ij})^2}{|E|}}$$

where E is the set of retained evaluation data.

In this project, we evaluate three models, namely:

- k-nearest neighbours (KNN), a simple neighbourhood model

- singular value decomposition (SVD), a latent factor model that accounts for user bias

- neural collaborative filtering (NCF), a latent factor model that represents features with neural network weights

Originally, we proposed three models KNN, SVD and SVD++ (a SVD model with time features). Due to the high similarity between SVD and SVD++, we proposed the use of NCF instead, while the additional time effect bias in SVD++ is left out as a to-do.

1.3. Notations

In this report, we denote variables about the dataset as follows:

- m: number of users

- n: number of movies

- N: number of ratings

- r_{ij}: rating of user i on movie j

2. Data preprocessing

Due to performance issues with CSV parsing, we wrote a separate program 'preprocess-dataset', which reformats the input files into a numpy saved array which contains the following columns derived from the original dataset:

- User ID

- Movie ID

- Rating value

- Year of rating

- Day of year of rating

- Day of week of rating

- Whether the rating is probed data

During runtime, the data are represented in compressed sparse row (CSR) format for a sparse $m \times n$ matrix.

3. KNN

KNN is an example of neighbourhood models. It seeks to provide recommendations based on the behaviour of similar users. In KNN model, we assume that similar users will give close ratings on similar movies. To predict the rating $\hat{r}_{ij}$ of user i on movie j ($1 \le i \le m, 2 \le j \le n$), the k most similar users ($n_1, \ldots, n_k$) to user i are computed based on similar choices of ratings, and $\hat{r}_{ij}$ is estimated based on the known values $r_{n_1 j}, \ldots, r_{n_k j}$.

3.1. Algorithm

The training algorithm involves the following steps:

The algorithm first finds q, the set of q users with the highest number of ratings. The distance $d(i, j)$ for $i \in [1, m], j \in q \setminus \{i\}$ is then computed, and the neighbourhood $K_i \subset q$ of each user i is evaluated as the k users with the lowest distance. The prediction of the rating r_{ij} is computed by aggregating the neighbourhood ratings with $A(\{r_{xj} : x \in K_i\})$. The memory complexity of these operations (except d and A) is $O(qn)$.

In this KNN implementation, there are four hyperparameters to be considered, namely neighbourhood size k, neighbourhood candidate size q, distance function d and aggregation function A.

3.1.1 Neighbourhood size k

k refers to the number of nearest neighbours to be included in recommendation system.

A large value of k would reduce accuracy as users with lower similarity are selected. On the other hand, a small value of k would be biased over the choice of the most similar user. We have tested with different values of k with different values of the other hyperparameters.

As a baseline model, we also set $k = m$, i.e. to evaluate the RMSE by taking the plain arithmetic mean of all users.

3.1.2 Neighbourhood candidate size q

q means the number of neighbour candidate the recommendation system considered. We tried different values of q, which led to different sets of k-nearest neighbours being selected each time.

3.1.3 Distance function d

The distance function $d : \mathbb{R}^n \times \mathbb{R}^n \to [0, 1]$ observes the deviation in interests between two different users. $d(i, j) = 0$ iff $i = j$, and $d(i, j) = d(j, i) \forall i = j$. Although $d(i, j) < d(i, k)$ implies that j is more similar to i than k is, it is not necessary that d follows triangular inequality.

Cosine The cosine distance is defined as

$$d(\mathbf{x}, \mathbf{y}) = 2 \frac{(\mathbf{x} - \mathbf{b}) \cdot (\mathbf{y} - \mathbf{b})}{\|\mathbf{x} - \mathbf{b}\| \|\mathbf{y} - \mathbf{b}\|} - 1$$

where $\mathbf{b} = \frac{1}{n} \sum_{i=1}^{m} \mathbf{r}_{i\cdot}$, i.e. the mean rating of each movie. This metric emphasizes the difference between users who gave ratings better or worse than the average, e.g. the difference between 3 and 5 is more important than the difference between 1 and 3 when the average rating is 4. It can be thought of a metric to determine how "abnormal" a user is. The multiplier is to correct the distance range to $[0, 1]$.

p-norm The p-norm distance is defined as the p-norm of two vectors normalized by the maximum possible distance, i.e.

$$d(\mathbf{x}, \mathbf{y}) = \frac{(\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p}}{4n^{1/p}}$$

.

3.1.4 Aggregation function A

We computed both baseline and average aggregation functions.

3.2. Findings

3.2.1 Baseline model result

The RMSE value attained in baseline model is 1.0528.

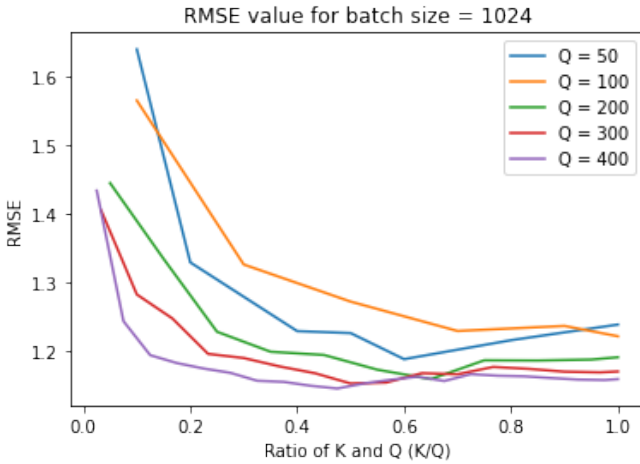### 3.2.2 Cosine distance model result

Figure 4. RMSE for different combination of q and k with cosine distance function

We can see the RMSE values decreased with the increasing q value in general. The curve with $q = 400$ has the smallest values for nearly all ratio of k and q. The curve with $q = 50$ however has larger values for most of the ratio

values than the curve with $q = 100$. All the curves have non-smooth logarithmic curves with fluctuations between ratio = 0.1 to ratio = around 0.7. The lowest RMSE value attained with the cosine distance function is 1.1455 with ratio = 0.475 ($q = 400$ and $k = 190$). For the other values of q, when $q = 300$, the lowest value found in ratio = 0.5 ($k = 150$). When $q = 200$, the lowest value belonged to ratio = 0.65 ($k = 130$). When $q = 100$, the lowest value attained at ratio = 1 ($k = 100$). When $q = 50$, the lowest belonged to ratio = 0.6 ($k = 30$). To conclude, the ratios for the lowest value of RMSE for each curves varied.

3.2.3 p-norm

We tested 1-norm (taxicab) and 2-norm (Euclidean) with smaller data sizes and discovered that they have significantly worse results (RMSE $1.176, 1.174$) compared to cosine metric with the same dataset (RMSE 1.106).

Since the computation of p-norm requires allocating a tensor of nqb integers (where b is the batch size), which implies a large memory size is required. A smaller choice of b must be selected, which greatly reduces the speed of model training. Considering the dissatisfactory RMSE of p-norm under smaller data sizes, we do not evaluate them for the full dataset.

4. SVD

4.1. Background

SVD assumes that ratings are modelled with the following formula:

$$r_{ij} = \mu + a_i + b_j + \sum_{x=1}^{k} u_{ix} v_{jx} + \varepsilon \text{ for } i \in [1, m], j \in [1, n]$$

μ is the global baseline score. a_i and b_j are the user and movie bias, which assumes that users and movies tend to have their own drift from the baseline ratings. In terms of psychology, this can be interpreted as certain users with more conservative with their ratings or certain movies that result in poor impression. These user/movie bias values are global to all movies/users, so the model separates them from the latent factors. The model also assumes that movies can be assessed with up to k metrics, which users have various interests about. This is represented by the latent factors u_{ix} and v_{jx}.

The algorithm computes μ as the mean of all ratings $\frac{1}{mn} \sum_{i,j} r_{ij}$, and the bias values as user and movie rating means $\frac{1}{n} \sum_j (r_{ij} - \mu), \frac{1}{m} \sum_i (r_{ij} - \mu)$. The latent factors are initialized with the distribution $\text{N}(0, \frac{\sigma}{200})$, where σ is the standard deviation of all ratings and 200 is an arbitrarily selected number based on actual trials. This ratio is not

exposed as a hyperparameter since descent parameters are random. The reason for using a normal distribution instead of a constant value is that the gradient descent ofrmula suffers from division by zero if the parameters are initialized as zero.

The gradient descent formula is derived to be

$$\begin{cases} u_{ik} & \leftarrow u_{ik} + \alpha \left(\sqrt{s_i} \varepsilon_{ij} v_{jk} - \lambda u_{ik} \right) \\ v_{jk} & \leftarrow v_{jk} + \alpha \left(\sqrt{s_j} \varepsilon_{ij} u_{ik} - \lambda v_{jk} \right) \end{cases}$$

where s_i, s_j are the number of ratings for the user/movie.

4.1.1 Learning rate α

The learning rate determines the amount of gradient descent to apply in each epoch. Since each user/movie has very different number of ratings, the descent in each iteration is divided by the square root of the number of ratings. As such, the learning rates selected are slightly higher than usual.

4.1.2 Regularization λ

Regularization is applied to avoid overfitting in the model. It subtracts the descent with a proportion of the original latent factors.

4.1.3 Rank of factorization k

The model assumes that movies and user interests are matched using a discrete number of metrics. This is modelled using different ranks of factorization. It is expected that each rank represents some linear combination of the actual metrics. A small choice of K limits the best possible accuracy of the model, while a large choice of k tends to cause overfitting. In fact, in preliminary tests with $m = 10000, n = 1000$, we discovered that choosing $k = 100$ brought excellent train accuracy but very poor test RMSE.

4.1.4 Number of epochs ξ

In each epoch, all training data are cycled through once and the total training RMSE is reported.

Due to time constraints, we only train up to $\xi = 15$ or $\xi = 20$ for each model. However, it was tested on smaller tests that an infinite number of epochs shall converge in RMSE.

4.1.5 Batch size β

Due to performance reasons, ε_{ij} is recomputed in batches (as an instance of minibatch stochastic gradient descent). The batch size is selected as 1048576 for all models evaluated.

4.2. Findings

5. NCF

Among all collaborative filtering techniques, matrix factorization (MF) become a standard approach to the model-based recommendation. Many research teams have put their efforts on enhancing it to break the accuracy record.

However, it has several limitations such as the cold-start problem – how the system provide recommendations to new users, how to recommend when new item is added without feature vector or embedding [4]. Also, the use of dot product for recommending popular items will lead to the tendency of recommending popular items to those users without specific user interests. With the use of inner product of User and item feature embeddings, MF limits the capture of the complex relations of the users and items interactions [2].

In this assignment, we aim to implement the new approach, Neural Collaborative Filtering (NCF), proposed by He team in 2017 [2]. They suggested that their design of deep neural network can overcome the limitation of MF.

5.1. Model structure

This is the model proposed by He team in 2017 [2]. Their proposed model uses a combination of matrix factorization and multi-layer perceptron techniques, and the hidden layers from the two techniques are combined by a fully-connected layer. However, their paper mainly focuses on the prediction of *implicit user feedback*, that is, they are predicting the values of y_{ui} where $y_{ui} = 1$ when the obervation (user u, item i) is observed, and 0 when the observation (user u, item i) is not observed. Thus, at the final stage of their proposed model, the output of the fully-connected layer is passed into a sigmoid function, such that the model only outputs values within $[0, 1]$.

The problem of predicting implicit feedback is different from the problem we wish to solve in the Netflix dataset, because in the Netflix problem we wish to predict actual ratings from $[1, 5]$. Nevertheless, we still think that He's paper provides a good starting point - We modify He's model by removing the final sigmoid function and changing the loss functions used directly, such that the neural network solves a regression problem instead of the original binary classification problem.

The model proposed by He is split into 3 parts - Generalized Matrix Factorization, Multi-Layer Perceptron, and Neural matrix factorization. In fact, the third part is a combination of the previous two parts. For this part of our project, we will follow the same workflow as He and implement the 3 parts separately.
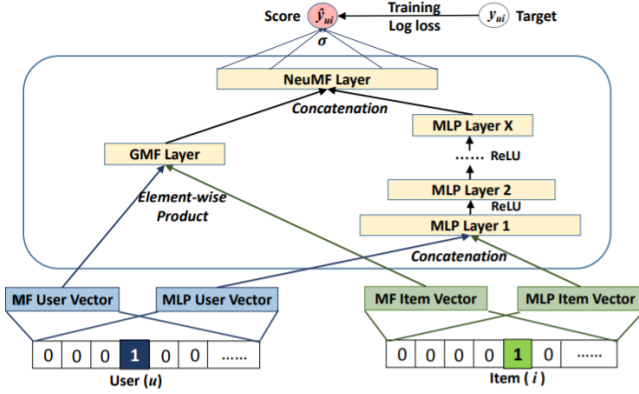
Figure 5. Neural Collaborative Filtering Model Design [2]

5.1.1 Model - Input Layer

The input of the model are the user ID and item ID transformed to a binarized vector with one-hot encoding.

5.1.2 Model - Embedding Layer

For each user and each movie, we train two sets of embedding latent vectors. This is because one set of latent vectors will be passed into the itemwise multiplication part of the model (in Hes paper, it is called Generalized Matrix Factorization (GMF)), and the other set of latent vectors will be passed into the neural network part of the model (in Hes paper, it is called Multi-Layer Perceptron (MLP)).

5.1.3 Model - Generalized Matrix Factorization (GMF)

In this part of the model, the user latent vector and item latent vector are combined via itemwise multiplication, similar to that in the SVD matrix factorization model. The output of this part is another vector with the same size at the latent vectors and is passed on to the final output layer of the model.

5.1.4 Model - Multi-Layer Perceptron (MLP)

In this part of the model, the user latent vector and item latent vector are first concatenated. Afterwards, the whole concatenated vector is passed through several layers of fully connected hidden linear layers, using the ReLU activation function. the output of this part is a vector dependng on the output size of the last hidden layer and is passed on to the final output layer of the model.

5.1.5 Model - Output Layer

The final output layer takes in the vectors produced by the GMF and MLP parts of the model and concatenates them.

The output of this part, which is the final output of the whole model, is a weighted sum of the vector elements, where the weights are model parameters to be learned. The final output layer produces a predicted score $\hat{y_{ui}}$.

5.1.6 Model - Formula

The formula of the GMF is as follow:

$$\hat{y_{ij}} = f(P^T v_i Q^T v_j | P, Q, \Theta_f)$$

The variables with m users and n items are denoting as:

- $P \in \mathbb{R}^{(m \times K)}$ latent factor matrix for users from embedding layer.

- $Q \in \mathbb{R}^{(n \times K)}$ latent factor matrix for items from embedding layer.

- f interaction function.

- Θ_f model parameters of the interaction function f

For MLP f, it can be formulated as:

$$f(P^T v_i, Q^T v_j) = \phi_{output}(\phi_X(...\phi_2(\phi_1(P^T v_i, Q^T v_j))...))$$

The mapping functions are as follow:

- ϕ_{output}: the mapping function for output layer

- ϕ_x the activation function for the xth hidden neural FC layer. x is the number of the layers

5.2. Technical Details

The movies are loaded into a list with columns of Movie ID, User ID and Rating. Then, it will be fed into the test loader with batch size.

We select Adadelta as our optimizer. To have a better performance on the converge of loss function,we implement a scheduler with different combinations of hyperparameters of scheduler.

5.3. Hyperparameters selection for scheduler

There are two hyperparameters, namely

- step size

- γ

5.4. Hyperparameters selection for model

In this MLP model, there are five hyperparameters to be selected, namely

- Learning Rate (α)

- Regularization (λ)

- Rank of factorization (k)

- Number of epoch (x)

- Batch Size (N)

5.4.1 Learning Rate (α)

The learning rate indicates the speed at which the model learns. It controls how much to change the modal to respond the estimate error each time when the model weights are updated.

Rate with too small value may cause a long training time while a large value may allow the modal learning too fast to have less accurate or even not meaningful result.

5.4.2 Rank of factorization (k)

Intuitively, each rank represents some characteristic of a user or a movie. Users and movies that interact strongly on a rank would be affected.

A large value of k increases the training time, memory consumption and probably the chance of overfitting, while a low value of k reduces the representativeness of the model and hence reduced accuracy.

5.4.3 Number of epoch (x)

The number of epochs defines the number of times that the modal will work through the whole training dataset. Since we already report the training score for each epoch, it is adjusted to a value such that the training score appears to converge negligibly.

5.4.4 Batch size (N)

The batch size defines the number of samples to be passed through before the update of model parameters.

The small batch size usually can give a better performance since the model can start learning with a small subset and have more cycles to improve.

However, the smaller batch size costs the slower learning and hence more training time.

Taking these factors into consideration, we decided that the batch size 65535 shall be used for all models, .

5.5. Predictive test set score (RMSE)

The model is evaluated by computing the RMSE between predicted and actual rating values:

$$\text{RMSE} = \sqrt{\sum_{(i,j) \in E} \frac{(\hat{r}_{ij} - r_{ij})^2}{|E|}}$$

where E is the set of retained evaluation data.

5.6. Findings

The following table exhaustively lists our test results. Train RMSE and Test RMSE refer to the RMSE value calculated for the training dataset and testing dataset respectively.
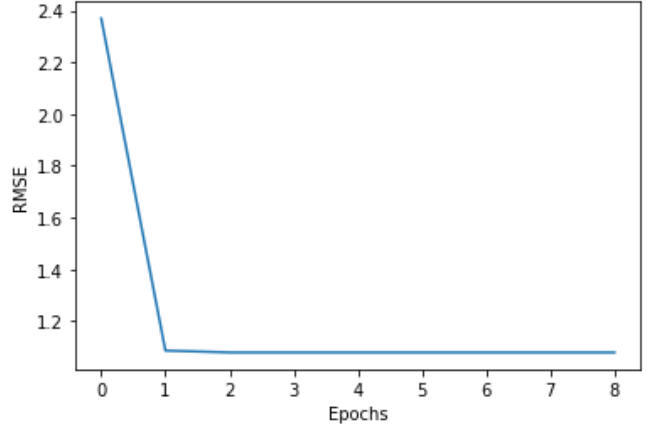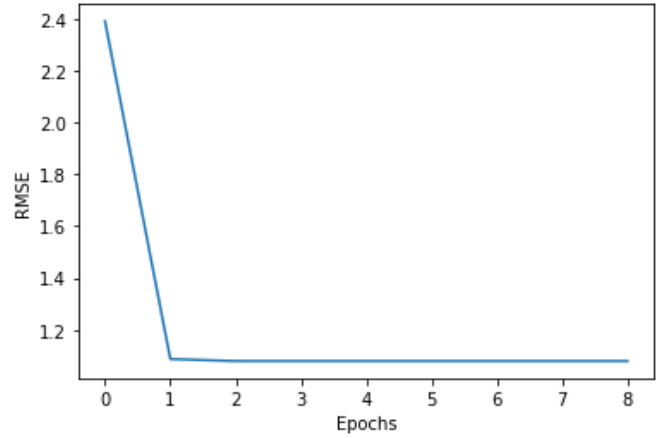

Figure 6. RMSE graph for model GMF(1)

Figure 7. RMSE graph for model GMF(2)

Model	k	α	γ	Epochs	Train RMSE	Test RMSE
GMF(1)	100	1	0.7	9	1.16384	1.08611
GMF(2)	300	1	0.7	9	1.16491	1.08696
GMF(3)	100	0.1	0.9	9	1.16536	1.08816
MLP(1)	100	1	0.7	8	1.11277	1.07873
MLP(2)	300	1	0.7	7	1.06405	1.11101
MLP(3)	100	0.1	0.9	7	1.14490	1.08372
NCF(1)	100	1	0.7	9	1.16384	1.08611
NCF(2)	300	1	0.7	9	1.16491	1.08696
NCF(3)	100	0.1	0.9	9	1.16536	1.08816

For GMF, when comparing the results of tuning the value of rank from 100 to 300, we can see the RMSE curves have a similar shape in general. After epoch 0, both MSE values drop significantly by around 4.5 units and then the values remain constant. The shape becomes smoother than previous results by tuning learning rate to 0.1 and γ to 0.9.

For MLP, when raising the rank from 100 to 300, we observe that the train RMSE decreases but the test RMSE increases. This may be due to overfitting and that an embedding with length 300 is too much for the MLP model. The
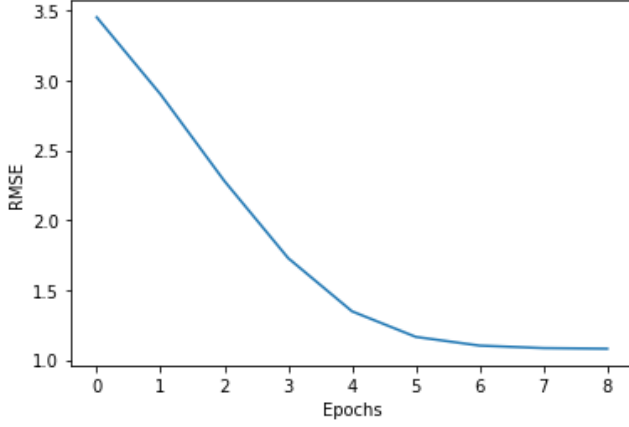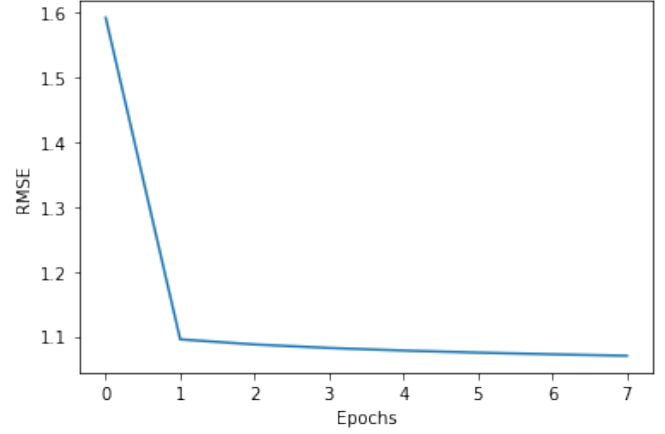
Figure 8. RMSE graph for model GMF(3)
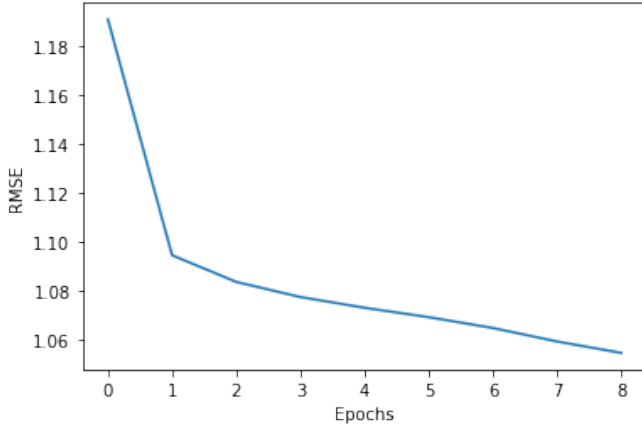

Figure 11. RMSE graph for model MLP(3)

Figure 9. RMSE graph for model MLP(1)

Figure 10. RMSE graph for model MLP(2)
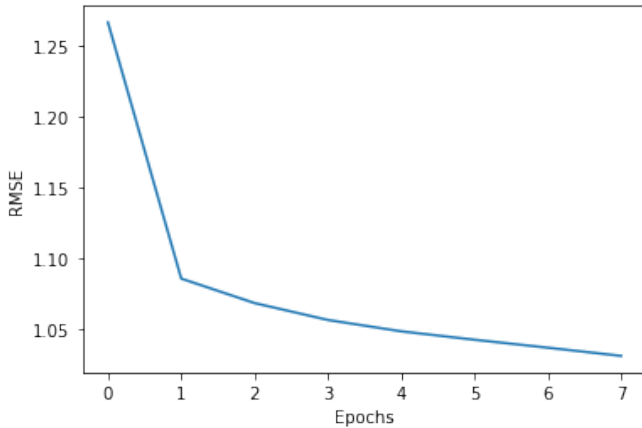
References

[1] Robert M Bell, Yehuda Koren, and Chris Volinsky. The bellkor 2008 solution to the netflix prize. *Statistics Research Department at AT&T Research*, 1(1), 2008. 2

[2] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017. 5, 6

[3] Y. Koren. Factorization meets the neighborhood: a multi-faceted collaborative filtering model. *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2008. 2

[4] Ke Zhou, Shuang-Hong Yang, and Hongyuan Zha. Functional matrix factorizations for cold-start recommendation. In *Proceedings of the 34th international ACM SIGIR conference on research and development in information retrieval*, SIGIR '11, pages 315–324. ACM, 2011. 5

shapes of the training curves are similar, with a large drop after the epoch 0. The drop becomes sharper after tuning the learning rate to 0.1 and γ to 0.9.

6. Conclusion