華南理工大學

South China University of Technology

# The Experiment Report of Deep Learning

**SCHOOL:** SCHOOL OF SOFTWARE ENGINEERING

**SUBJECT:** SOFTWARE ENGINEERING

| Author & Student ID | |
|---|---|
| Yuxuan Gao | 201720145075 |
| Yiqun Wu | 201720145082 |
| Ran Yang | 201721046128 |

Supervisor:
Qingyao Wu

Grade:

Graduate

December 26, 2017

# Recommender System Based on Matrix Decomposition

**Abstract—this experiment applies stochastic gradient descent (SGD) on recommend system based on Matrix Decomposition. The influence of the number of potential features and learning rate in the experiment is also discussed. Finally graphs of loss with the number of iterations in validation dataset and a comparison of different number of potential features and learning rate are presented. This experiment shows the apply of stochastic gradient descent (SGD) in recommend system based on Matrix Decomposition.**

## I. Introduction

It is probably no need to say that there is too much information on the Web nowadays. Search engines help us a little bit. What is better is to have something interesting recommended to us automatically without asking. Indeed, from as simple as a list of the most popular bookmarks on Delicious, to some more personalized recommendations we received on Amazon, we are usually offered recommendations on the Web.

Recommendations can be generated by a wide range of algorithms. While user-based or item-based collaborative filtering methods are simple and intuitive, matrix factorization techniques are usually more effective because they allow us to discover the latent features underlying the interactions between users and items. Of course, matrix factorization is simply a mathematical tool for playing around with matrices, and is therefore applicable in many scenarios where one would like to find out something hidden under the data.

In this experiment, we will firstly introduce the mathematics of matrix factorization, and then we will present a simple implementation in Python. We will deal with the user ratings (an integer score from the range of 1 to 5) of movie in a recommendation system.

The motivation of this experiment is as follows:

1. Explore the construction of recommended system.
2. Understand the principle of matrix decomposition.
3. Be familiar to the use of gradient descent
4. Construct a recommendation system under small-scale dataset, cultivate engineering ability.

## II. Methods and Theory

### A. Basic Ideas of matrix factorization

Matrix factorization [2] is to factorize a matrix, i.e. to find out two (or more) matrices such that when multiply them will get back the original matrix. From an application point of view, matrix factorization can be used to discover latent features underlying the interactions between two different kinds of entities. (Matrix factorization can consider more than two kinds of entities and will be dealing with tensor factorization, which would be more complicated.) And one obvious application is to predict ratings in collaborative filtering.

In a recommendation system such as Netflix or MovieLens, there is a group of users and a set of items (movies for the above two systems). Given that each users have rated some items in the system, we would like to predict how the users would rate the items that they have not yet rated, such that we can make recommendations to the users. In this case, all the information we have about the existing ratings can be represented in a matrix.

The intuition behind using matrix factorization to solve this problem is that there should be some latent features that determine how a user rates an item. For example, two users would give high ratings to a certain movie if they both like the actors/actresses of the movie, or if the movie is an action movie, which is a genre preferred by both users. Hence, if we can discover these latent features, we should be able to predict a rating with respect to a certain user and a certain item, because the features associated with the user should match with the features associated with the item.

In trying to discover the different features, we also make the assumption that the number of features would be smaller than the number of users and the number of items. It should not be difficult to understand this assumption because clearly it would not be reasonable to assume that each user is associated with a unique feature (although this is not impossible). And anyway if this is the case there would be no point in making recommendations, because each of these users would not be interested in the items rated by other users. Similarly, the same argument applies to the items.

### B. Matrix factorization based on stochastic gradient descent

Assume a set U of users, and a set D of items. Let R of size |U| and |D| be the matrix that contains all the ratings that the users have assigned to the items. Also, assume that we would like to discover K latent features. The task is to find two metrics P (a |U| * K matrix) and Q (a |D| * K matrix) such that their product approximates R:

$$R_{m\times n} \approx P_{M\times K} \times Q_{K\times N}$$

In this way, each row of P would represent the strength of the associations between a user and the features. Similarly, each row of Q would represent the strength of the associations between an item and the features. To get the prediction of a rating of an item by, we can calculate the dot product of the two vectors corresponding to u_i and d_j:

$$r_{ij} = p_i q_j = \sum_{K=1}^{K} p_{ik} q_{kj}$$

One way to approach to obtain P and Q is firstly initializing two matrices with some values, and then try to minimize this difference iteratively. Such a method is called gradient descent, aiming at finding a local minimum of the difference [1].

The difference here, usually called the error between the estimated rating and the real rating, can be calculated by the following equation for each user-item pair:

$$e_{ij} = (r_{ij} - \sum_{K=1}^{K} p_{ik} q_{kj})^2$$

Consider the squared error because the estimated rating can be either higher or lower than the real rating.
A common extension to this basic algorithm is to introduce regularization to avoid overfitting. This is done by adding a parameter β and modify the squared error as follows:

$$e_{ij} = (r_{ij} - \sum_{K=1}^{K} p_{ik} q_{kj})^2 + \frac{\beta}{2} \sum_{K=1}^{K} (\|P\|^2 + \|Q\|^2)$$

Differentiate the above equation with respect to these two variables separately:
In other words, the new parameter β is used to control the magnitudes of the user-feature and item-feature vectors such that P and Q would give a good approximation of R without having to contain large numbers.
Derivatives can be described by the following formulas [3]:

$$p'_{ik} = p_{ik} + \alpha \frac{\partial e_{ij}^2}{\partial p_{ik}} = p_{ik} + \alpha(2 e_{ij} q_{kj} - \beta p_{ik})$$

$$q'_{kj} = q_{kj} + \alpha \frac{\partial e_{ij}^2}{\partial q_{kj}} = q_{kj} + \alpha(2 e_{ij} p_{ik} - \beta q_{kj})$$

### *C. Stochastic gradient descent (SGD)*

SGD works similar as GD (gradient descent), but more quickly by estimating gradient from a few examples at a time

**Algorithm1: SGD**

```
1  Initialize parameter $w$ and learning rate $\eta$
2  while an approximate minimum is not
   obtained do:
3      Randomly select an example i in
       the training set
4      $w = w - \eta \nabla L_i(w)$
5  end
```

## III. Experiment

This experiment Utilize MovieLens-100k dataset. MovieLens data sets were collected by the GroupLens Research Project at the University of Minnesota. This data set consists of 100,000 ratings (1-5) from 943 users on 1682 movies.Each user has rated at least 20 movies. The data was collected through the MovieLens website (movielens.umn.edu) during the seven-month period from September 19th, 1997 through April 22nd, 1998. This data has been cleaned up: users who had less than 20 ratings or did not have complete demographic information were removed from this data set.
u.data consists 10,000 comments from 943 users out of 1682 movies. At least, each user comment 20 videos. Users and movies are numbered consecutively from number 1 respectively. The data is sorted randomly. u1.base / u1.test are train set and validation set respectively, separated from dataset u.data with proportion of 80% and 20%. It also make sense to train set and validation set from u1.base / u1.test to u5.base / u5.test.

The experiment steps of recommend system using stochastic gradient descent method (SGD):

1 Read the data set and divide it (use u1.base / u1.test to u5.base / u5.test directly). Populate the original scoring matrix against the raw data, and fill 0 for null values.
2 Initialize the user factor matrix and the item (movie) factor matrix, set the number of potential features k.
3 Determine the loss function and hyper parameter learning rate and the penalty factor.
4 Use the stochastic gradient descent method to decompose the sparse user score matrix, get the user factor matrix and item (movie) factor matrix:
    4.1 Select a sample from scoring matrix randomly;
    4.2 Calculate this sample's loss gradient of specific row(column) of user factor matrix and item factor matrix;
    4.3 Use SGD to update the specific row(column) of P and Q;
    4.4 Calculate the loss on the validation set, comparing with the loss of the previous iteration to determine if it has converged.
5 Repeat step 4 for several times, get a satisfactory user factor matrix P and an item factor matrix Q, Draw a loss curve with varying iterations.
6 The final score prediction matrix is obtained by multiplying the user factor matrix P and the transpose of the item factor matrix Q.

In this experiment, we set the number of potential features k as 5, and the learning rate α is 0.02, the penalty factor β is 0.1. We select 8000 samples to train the system totally. The experiment result is shown as the following figure:

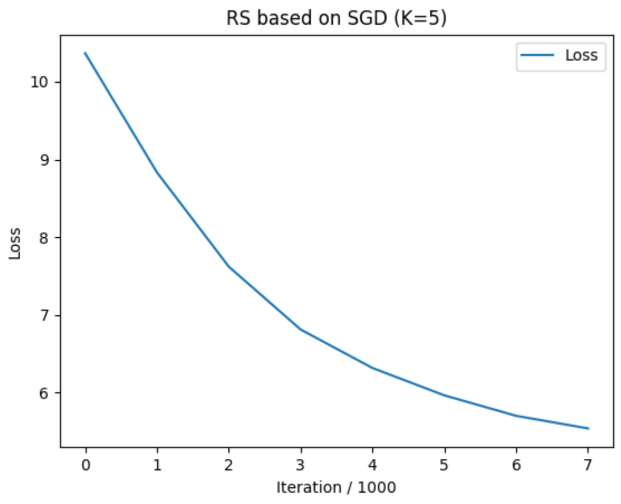


Figure1. RS based on SGD (k=5)

As show in figure1, the loss on the validation set decreases with the number of iterations. It reduces fast firstly and tend to level off.

To discuss the influence of the number of potential features k, we set the k as 2 and 7, at the same time we keep other factors remain the same and repeat the experiment. The experiment is shown as follows:

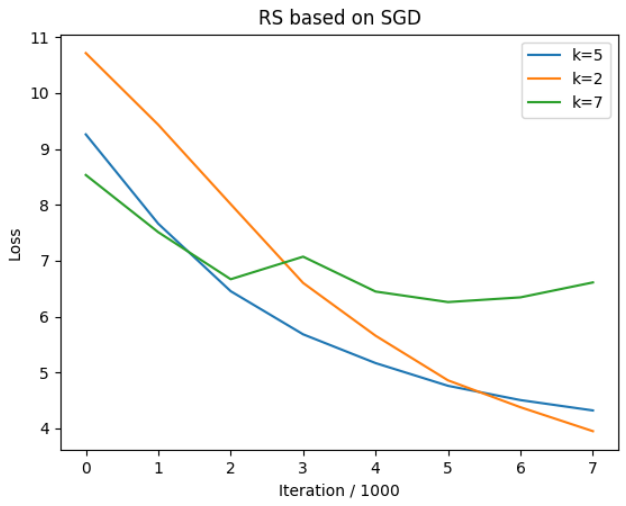
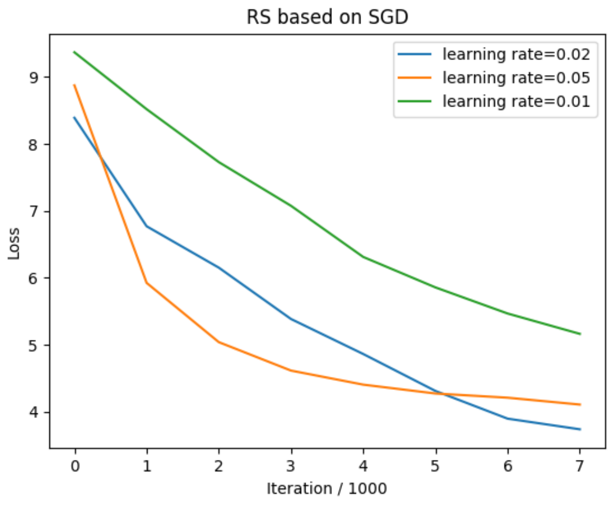


Figure2. Comparison of RS based on SGD (k=2, 5, 7)

As shown in the above graphs, generally, loss of different potential features k decade with the number of iterations and finally tend to converge to a value. As the effect of randomness, it will fluctuate on a value. The experiment result also shows that a suitable potential features k is helpful to make the loss converge quickly and reduce volatility.

To discuss the influence of the number of learning rate α, we set the α as 0.05, 0.02 and 0.01, at the same time we keep other factors remain the same and repeat the experiment. The experiment is shown as follows:



As shown in the above graphs, generally, loss of different suitable learning rate decade with the number of iterations and finally tend to converge to a value. As the effect of randomness, it will fluctuate on a value. The experiment result also shows that a suitable learning rate is helpful to make the loss converge quickly and reduce volatility.

## IV. CONCLUSION

In this experiment, we explore the construction of recommended system and have a better understanding of the principle of matrix decomposition. Also, we are familiar to the use of gradient descent. By constructing a recommendation system under small-scale dataset, we cultivate our engineering ability. From this experiment I also have a better understand of the principle of stochastic gradient descent. Through this apply of stochastic gradient descent on recommend system based on Matrix Decomposition, I also realize that some parameter have a great influence in the experiment result, and it's necessary to master the technique of tuning parameters.

## V. REFERENCE

[1] Xie, X., Tan, W., Fong, L. L., & Liang, Y. (2017, June). CuMF_SGD: Parallelized Stochastic Gradient Descent for Matrix Factorization on GPUs. In Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing (pp. 79-92). ACM.

[2] Tan, W., Cao, L., & Fong, L. (2016, May). Faster and cheaper: Parallelizing large-scale matrix factorization on gpus. In Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing (pp. 219-230). ACM.

[3] http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python/