Deep Learning Collaborative Filtering for Recommendation

Yeshu Li*
yli299@uic.edu
Department of Computer Science
University of Illinois at Chicago
Chicago, Illinois

Yang Hao*
yhao5@uic.edu
Department of Physics
University of Illinois at Chicago
Chicago, Illinois

ABSTRACT

Recommender systems have been a popular research topic and widely adopted in industry. Motivated by lack of consideration about the latent factor underlying user-item interactions, researchers put considerable efforts in studying collaborative filtering methods that model dependency of distinct users and items. Matrix factorization is the first attempt to find low-dimensional latent features of objects. Recent deep learning models for collaborative filtering recommendation further introduce natural language processing techniques such as semantic analysis and word embedding to take advantage of the rich textual information in addition to a raw rating matrix. In this paper, we study and compare three state-of-the-art collaborative filtering recommender systems including one matrix factorization approach and two deep learning models. Analysis and discussion on the experiment results are followed by a brief conclusion and description about potential extensions.

CCS CONCEPTS

• **Computing methodologies → Information extraction**.

KEYWORDS

information retrieval, recommender systems, machine learning, deep learning, matrix factorization, collaborative filtering

ACM Reference Format:
Yeshu Li and Yang Hao. 2018. Deep Learning Collaborative Filtering for Recommendation. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Recommendation algorithms have become important and widely utilized by a wide range of industries. Recommendation systems not only help customers get valuable information and increase revenues by increasing the Customer Stickiness, but also cut off the marketing cost. Many retail giants like Amazon, Best Buy and Walmart credit recommendation with an incredible increase in total revenue. There are two basic recommender algorithms:

- Content-based systems focus on properties of items and users. Similarity of items and users is weighted by the similarity in their properties. For example, online store may suggest users to buy items with similar features to an item similar users mostly purchased.
- Collaborative-filtering systems focus on the relationship between users and items. Similarity of items is determined by the similarity of the ratings of those items by the users who have rated both items. Standard collaborative filtering benchmarks solve the matrix completion task by taking into account the collective interaction data to predict future ratings or purchases.

In our project, we implemented three different recommendation algorithms to predict the ratings of users to items in three public datasets (MovieLens 10M, Douban Movie and Yahoo Music).

The first recommender is the popular Factorization Machines (FMs) [22] using stochastic gradient descent with regularization. The interaction among variables is mapped to a low dimensional space. It belongs to the collaborative filtering systems and performs well on huge sparse datasets. The performance of FM is used as the state-of-art model. In our project, the FM model is based on an open-source software Setffen's libFM.

The second recommender is the Graph Convolutional Matrix Completion (GCMC) model based on the paper by Rianne et al. [2], which is a content-based system on bipartite interaction graph between user and item nodes, And content information as users and items' features are incorporated into nodes by graph auto-encoder. The recommender is trained on datasets through stochastic mini-batching.

And the last recommendation system is the Factorized Exchangeable Autoencoder (FEA) [16] consisting of an encoder and a decoder. Deep learning is applied to model interactions across sets of objects as a matrix with an exchangeability property. The learning model is designed as permutation equivariant (PE): parameter-sharing approach for achieving it. The set of prediction labels based on the encoder is not changed by permuting rows or columns. Also unlike these other approaches, FEA has a number of parameters independent of the size of the matrix to be completed. And it is also able to do transfer learning tasks by extrapolating from one dataset to other datasets ratings.

In what follows, we begin by presenting some of the most successful related work about recommendation systems in Section 2. Section 3 demonstrates the technical details and philosophy of three feasible and powerful recommendation algorithms. Applied datesets, experimental settings and results on several public datasets are shown in Section 4, with conclusion and future research directions discussed in Section 5.

*Both authors contributed equally to this research.

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

2 RELATED WORK

The literature in matrix factorization and completion is vast. The classical approach to solving the matrix completion problem is to find some low rank (or sparse) approximation that minimizes a reconstruction loss for the observed ratings (see e.g., Candes and Recht [3], 2009; Koren et al. [18], 2009; Mnih and Salakhutdinov, [20], 2008). They make predictions about users and items observed during training. And there are also some deep factorization and other collaborative filtering techniques (e.g., Salakhutdinov et al. [24], 2007; Deng [6] et al.; Sedhain et al. [26], 2015; Wang [30] et al., 2015; Li et al. [19], 2015; Zheng et al. [32], 2016; Dziugaite and Roy [10], 2015).

FMs can mimic most of factorization models just by using the right input data (e.g. feature vector $\mathbf{x}$), including matrix factorization, svd++, PITF, FPMC, and standard models like MF, PaRAFAC etc. The implementation for factorization machines also provides interface to select stochastic gradient descent [22], alternating least squares optimization [23] and Bayesian inference using Markov Chain Monte Carlo [11] [23] to boost the performance.

Credited to edge weights in bipartite graph from user-item rating data, convolutional neural networks deep learning algorithms are rapidly developed. More literature [31] [12] [28] claimed user- or item-based auto-encoders which only had user or item embeddings. Thus those encoders can be considered as state-of-the-art collaborative filtering models and also as a special case of the graph auto-encoder model mentioned in this project. The first such model named AutoRec was developed by Sedhain et al [26]. The CF-NADE algorithm by Zheng et al. [32] has messages in the user-based setting passing from items to users. And item-based case works in the opposite way. CF-NADE imposes a random ordering on nodes of interaction graph, and splits incoming messages into two sets via a random cut.

Inspired by the work by Yang et al. [27] that extends factorization-style approaches to the inductive setting (where predictions can be made on unseen users and items), which is different from the traditional MF models because additional side information is embedded as user and item features. A few methods have been published by (Berg et al. [2], 2017; Monti et al. [21], 2017) showing their effort in predicting edge weights in bipartite graphs. And some work about convolutional neural networks to graph-structured data (Scarselli et al. [25], 2009; Bruna et al. [7], 2014; Duvenaud et al. [9], 2015; Defferrard et al. [5], 2016; Kipf and Welling [17], 2016; Hamilton et al. [14], 2017) has seen surge of research. The deep learning graph convolutional matrix completion algorithm is inspired by the inductive setting and convolutional neural networks to build a graph auto-encoder framework based on differentiable message passing on the bipartite interaction graph. Parameter sharing in graph convolution is also been covered in the way of exchangeable matrices.

3 METHODS

In this section, we make a detailed description of the technical details of three methods adopted in this paper.

3.1 Factorization Machines

In a general machine learning task, the goal is usually to find the optimal function from a given hypothesis space to minimize the empirical risk with regard to a set of training data sampled from the target true distribution. In a supervised learning setting, given a training set $D = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$, the empirical risk minimization (ERM) is formulated as:

$$\hat{h} = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(h(\mathbf{x}^{(i)}), y^{(i)}), \tag{1}$$

where $\mathcal{L}$ is some loss function, $\mathcal{H}$ is a hypothesis space, $h : \mathbb{R}^m \rightarrow Y$ denotes a function mapping from a real-valued n-dimensional feature vector to the target domain Y. Depending on the range of the target domain Y, the machine learning task becomes regression if $Y = \mathbb{R}$, or classification if $Y = \{+1, -1\}$. A specific loss function is designed to fit different learning settings with different models. For example, the hinge loss is notably adopted in support vector machines (SVMs):

$$\mathcal{L}(y) = \max(0, 1 - ty). \tag{2}$$

In matrix factorization approaches for recommender systems, a scoring function in a ranked form is often useful. For instance, a pair of samples $(\mathbf{x}^{(A)}, \mathbf{x}^{(B)}) \in D$ enforces that $\mathbf{x}^{(A)}$ should be put ahead of $\mathbf{x}^{(B)}$ in the final structured rank prediction.The training dataset in our project consists of a set of tuples, $(u, i, t, r) \in S$, where $u \in U$ denotes a user, $i \in I$ represents an item, $t \in \mathbb{R}$ stands for a timestamp and $r \in \{1, 2, 3, 4, 5\}$ is a standard 5-star rating domain we are interested in. The learning problem for recommender systems is thus to find the optimal function $h^* : U \times I \times \mathbb{R} \rightarrow \{1, 2, 3, 4, 5\}$ to predict a rating score r for an unseen tuple of (u, i, t).

In a practical recommender systems setting, the feature vector $\mathbf{x}^{(i)}$ is usually sparse. In other words, let $m(\mathbf{x}^{(i)})$ represent the number of non-zero elements in $\mathbf{x}^{(i)}$, and $\overline{m}_D$ denote the average number of non-zero elements in domain D, for example:

$$\overline{m}_D = \frac{1}{n} \sum_{i=1}^{n} m(\mathbf{x}^{(i)}), \quad \mathbf{x}^{(i)} \in D. \tag{3}$$

There is significant sparsity, in which $\overline{m}_D \ll m$ in many real-world applications such as recommender systems, event transactions, text analysis, speech recognition etc.The underlying latent factor for objects therefore is sufficiently representable in a low-dimensional feature space. For instance, there are one-to-one correspondences between data instances and a discrete domain with a large number of categorical values.

For clarity, at first, we focus a quadratic factorization machine (FM) [22] as follows:

$$\hat{h}(\mathbf{x}; \theta) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j, \tag{4}$$

where the model parameters θ include:

$$w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^n, \mathbf{V} \in \mathbb{R}^{n \times k}, \tag{5}$$

and the dot product of two feature vectors is defined as:

$$\langle \mathbf{v}_i, \mathbf{v}_j \rangle = \sum_{f=1}^{k} v_{i,f} \cdot v_{j,f}. \tag{6}$$

Hence, $\mathbf{v}_i$ is the i-th row of the factorized feature matrix $\mathbf{V}$ and $k \in \mathbb{Z}^+$ is a predefined hyperparameter for the dimensionality of the latent space. In this way, w_0 takes care of global bias, w_i deals with the unary strength of x_i and $w_{i,j} = \langle \mathbf{v}_i, \mathbf{v}_j \rangle$ that models binary interaction between x_i and x_j. The last term about the interaction is the key point of FMs because it is feasible for us to model high order interactions under the sparsity assumption with this dot product form.

Generally speaking, assuming that $\mathbf{W}$ is positive definite and k is sufficiently large, an FM is able to model the interaction by decomposition:

$$\mathbf{W} = \mathbf{V} \cdot \mathbf{V}^T. \tag{7}$$

Thus, FMs break the independence of the interaction parameters by factorizing them. However, a small k is usually chosen in practice because of smaller space complexity, sparse data settings and better generalization.

At first glance, the time complexity for computing Equation 4 is $O(kn^2)$ because of the n^2 quadratic terms and the k element-wise dot product computation. Even so, this can be reduced to linear time in terms of k and n by reformulation, which leads the following lemma:

Lemma 1: The model equation of a factorization machine can be computed in linear time $O(kn)$.

Since the dot product term does not jointly depend upon both of the two variable indexed by i and j, the quadratic terms in Equation 4 can be written as follows:

$$\begin{aligned}
&\sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \\
=&\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j - \frac{1}{2} \sum_{i=1}^{n} \langle \mathbf{v}_i, \mathbf{v}_i \rangle x_i x_i \\
=&\frac{1}{2} \Big(\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{f=1}^{k} v_{i,f} v_{j,f} x_i x_j - \sum_{i=1}^{n} \sum_{f=1}^{k} v_{i,f} v_{i,f} x_i x_i \Big) \\
=&\frac{1}{2} \sum_{f=1}^{k} \Big((\sum_{i=1}^{n} v_{i,f} x_i)(\sum_{j=1}^{n} v_{j,f} x_j) - \sum_{i=1}^{n} v_{i,f}^2 x_i^2 \Big) \\
=&\frac{1}{2} \sum_{f=1}^{k} \Big((\sum_{i=1}^{n} v_{i,f} x_i)^2 - \sum_{i=1}^{n} (v_{i,f} x_i)^2 \Big).
\end{aligned}$$

Under the sparsity assumption that most of the entries in $\mathbf{V}$ are zeros, the complexity is further reduced to $O(k\overline{m}_D)$, and $\overline{m}_D = 2$ for a typical recommender system.

As discussed in the beginning, mean squared error can be adopted for the loss function $\mathcal{L}(h(\mathbf{x}^{(i)}), y^{(i)})$ if the prediction task is modeled as regression, hinge loss is used in binary classification, and discounted cumulative gain is useful in ranking prediction.

With a closed form function expression, an arbitrary gradient descent method can be adopted optimization such as stochastic gradient descent (SGD) for a variety of losses. Regardless of the loss functions leveraged, the partial derivative of $\hat{h}(\mathbf{x}; \theta)$ with respect to its parameters can be computed as follows:

$$\begin{aligned}
\frac{\partial}{\partial w_0} \hat{h}(\mathbf{x}; \theta) =& 1 \\
\frac{\partial}{\partial w_i} \hat{h}(\mathbf{x}; \theta) =& x_i \\
\frac{\partial}{\partial v_{i,f}} \hat{h}(\mathbf{x}; \theta) =& x_i \sum_{j=1}^{n} v_{j,f} x_j - v_{i,f} x_i^2.
\end{aligned}$$

By precomputing the terms $\sum_{j=1}^{n} v_{j,f} x_j$, the time complexity of gradient updates can be done in $O(k\overline{m}_D)$ on average.

Following from Equation 4, a generalized FM with degree d can be formulated as:

$$\hat{h}(\mathbf{x}; \theta) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{l=2}^{d} \sum_{i_1=1}^{n} \cdots \sum_{i_l=i_{l-1}+1}^{n} (\prod_{j=1}^{l} x_{i_j}) (\sum_{f=1}^{k_l} \prod_{j=1}^{l} v_{i_j,f}^{(l)}), \tag{8}$$

where the factorization parameters are:

$$\mathbf{V}^{(l)} \in \mathbb{R}^{n \times k_l}, k_l \in \mathbb{Z}^+. \tag{9}$$

This general form can be computed in linear time as well by following similar reformulation in Lemma 1.

3.2 Graph Convolutional Matrix Completion

A recommendation task can be regarded as a matrix completion task, which has a rating matrix $M \in \mathbb{R}^{+N_u \times N_v}$ whose entry $M_{i,j}$ is either an observed rating, or 0 for an unobserved rating, as N_u, N_v are the number of users and items respectively. The task is thus to predict the missing values in M.

In another perspective, matrix completion can be considered as link prediction in bipartite graph for users and items. Formally, the data is represented as an undirected graph $G = (\mathcal{W}, \mathcal{E}, \mathcal{R})$, where $\mathcal{W} = \mathcal{U} \cup \mathcal{V}$ consisting of user nodes $u_i \in \mathcal{U}, i \in \{1, \ldots, N_u\}$ and item nodes $v_j \in \mathcal{V}, j \in \{1, \ldots, N_v\}$. $\mathcal{R} = \{1, \ldots, R\}$ denotes a set of possible ratings and $(u_i, r, v_j) \in \mathcal{E}$ stands for an edge implying user u_i gives a rating r to item v_j. Early work on graph-based approaches to recommendation typically performs graph feature extraction and link prediction in a multi-stage and independent manner. However, recent study shows that training a graph auto-encode in an end-to-end learning setting or unsupervised setting yields promising recommendation results.

The graph auto-encoder model in this method is called Graph Convolutional Matrix Completion (GCMC) [2]. It makes up of a graph encoder and a decoder. The graph encoder is a function defined as:

$$Z = f(X, A) : \mathbb{R}^{N \times D} \times \mathbb{Z}^{N \times N} \rightarrow \mathbb{R}^{N \times E}, \tag{10}$$

where X is the input feature matrix and A represents the corresponding adjacency matrix with $Z = [z_1^T, \ldots, z_N^T]^T$ being the output embedding matrix. N, D, E denotes the number of nodes, the

input feature dimension and the output embedding dimension respectively. The graph decoder is defined as:

$$\check{A} = g(Z) : \mathbb{R}^{N\times E} \rightarrow \mathbb{R}^{N\times N}, \tag{11}$$

which takes as inputs the embedding matrix and predicts the entries $M_{i,j}$ based on z_i and z_j in rows in Z.

In a bipartite recommendation graph, the encoder can be formulated as:

$$[U, V] = f(X, M_1, \ldots, M_R), \tag{12}$$

where $M_r \in \{0, 1\}^{N_u\times N_v}$ is an indicator matrix for rating type $r \in \mathcal{R}$, in which $M_{r,i,j} = 1$ if $M_{i,j} = r$, and $M_{r,i,j} = 0$ otherwise. $U \in \mathbb{R}^{N_u\times E}$ and $V \in \mathbb{R}^{N_v\times E}$ are embedding matrices for users and items respectively. Similarly, the decoder model can be written as:

$$\check{M} = g(U, V) : \mathbb{R}^{N_u\times E} \times \mathbb{R}^{N_v\times E} \rightarrow \mathbb{R}^{N_u\times N_v}. \tag{13}$$

Hence, the graph auto-encoder could be trained end-to-end by minimizing the reconstruction loss between $\check{M}$ and the ground-truth rating matrix M. Root mean squared error and cross entropy are commonly used loss metrics.

In this paper, a graph encoder with weight sharing convolutional layers is adopted. Although the local operations only take into account the first-order immediate neighbor nodes, they are efficient and applied across all the locations in the same manner. The convolution is equivalent to one form of message passing. In this way, the transformation can be regarded as an edge type specific message from item j to user i:

$$\mu_{j\rightarrow i,r} = \frac{1}{c_{ij}} W_r x_j, \tag{14}$$

$$c_{ij} = \frac{1}{|\mathcal{N}_i|}, \tag{15}$$

where c_{ij} is a constant for normalization, $\mathcal{N}_i$ is the set of neighbors of node i, W_r is a parameter matrix for rating r, with x_j being the input feature vector. After message passing, the belief for node i can be obtained as follows:

$$h_i = \sigma\big[accum\big(\sum_{j\in\mathcal{N}_{i,1}} \mu_{j\rightarrow i,1}, \ldots, \sum_{j\in\mathcal{N}_{i,R}} \mu_{j\rightarrow i,R}\big)\big], \tag{16}$$

where $accum(\cdot)$ is an accumulation function such as concatenation, stacking or pooling while $\sigma(\cdot)$ is a placeholder for some activation function. The final embedding for user node i is obtained after passing h_i through a fully connected layer and another activation layer:

$$u_i = \sigma(W h_i). \tag{17}$$

The final embedding for item node v_j follows the similar formulation.

A bilinear graph decoder is considered here that regards different rating levels as different classes. Assume that the reconstructed rating between user i and item j is $\check{M}_{ij}$, the predicted rating distribution follows the form of a softmax function:

$$p(\check{M}_{ij} = r) = \frac{e^{u_i^T Q_r v_j}}{\sum_{s\in R} e^{u_i^T Q_s v_j}}, \tag{18}$$

where $Q_r \in \mathbb{R}^{E\times E}$ is a trainable parameter matrix. The final predicted rating is thus:

$$\check{M}_{ij} = g(u_i, v_j) = \mathbb{E}_{r\sim p(\check{M}_{ij})}[r] = \sum_{r\in R} r p(\check{M}_{ij} = r). \tag{19}$$

For model training, similar to cross-entropy, the objective loss function can be written as the sum of the negative log-likelihood of the predicted ratings:

$$\mathcal{L}(M, \check{M}) = -\sum_{i,j;\Omega_{ij}=1} \sum_{r=1}^{R} I[r = M_{ij}] \log p(\check{M}_{ij} = r), \tag{20}$$

where $I(\cdot)$ is the indicator function and $\Omega \in \mathbb{Z}^{N_u\times N_v}$ serves as a mask matrix in which $\Omega_{ij} = 1$ for observed ratings and $\Omega_{ij} = 0$ otherwise. During optimization, a node dropout approach in which all the outgoing messages of a node are randomly discarded for robustness, and a mini-batch training strategy sampling a subset of interactions between users and items is adopted for efficiency.

Under the sparsity assumption, the implementation in practice could compute the embeddings in sparse matrix multiplication form:

$$\begin{bmatrix} U \\ V \end{bmatrix} = f(X, M_1, \ldots, M_R) = \sigma\Big(\begin{bmatrix} H_u \\ H_v \end{bmatrix} W^T\Big), \tag{21}$$

$$\begin{bmatrix} H_u \\ H_v \end{bmatrix} = \sigma\Big(\sum_{r=1}^{R} D^{-1} \mathcal{M}_r X W_r^T\Big), \tag{22}$$

$$\mathcal{M}_r = \begin{pmatrix} 0 & M_r \\ M_r^T & 0 \end{pmatrix}, \tag{23}$$

where D denotes the diagonal node degree matrix with diagonal entries $D_{ii} = |\mathcal{N}_i|$. Furthermore, with the help of sparsity, computing the encoded embedding is now $O(|\mathcal{E}|)$. Analogous multiplication equations can be obtained for the bilinear decoder.

When insufficient information in the input feature vectors becomes severe bottleneck of information flow, side information can be incorporated into the encoder forward passing:

$$u_i = \sigma(W h_i + W_2^f f_i), \tag{24}$$

$$f_i = \sigma(W_1^f x_i^f + b), \tag{25}$$

where x_i^f is the side information feature vector for node i while W_1^f, W_2^f and b are the corresponding trainable weight matrices and bias respectively.

The columns of the weight matrices W_r serve as nodes' latent factors for a specific rating r. However, because of the undue number of ratings for different rating levels, some weight sharing across ratings is necessary. The ordinal weight sharing includes more weight matrices for higher-rating matrices:

$$W_r = \sum_{s=1}^{r} T_s. \tag{26}$$

While the weight sharing in the graph decoder can be written as a linear combination of basis weight matrices P_s:

$$Q_r = \sum_{s=1}^{n_b} a_{rs} P_s, \tag{27}$$

where n_b is the number of basis with a_{rs} being learnable coefficients.

3.3 Factorized Exchangeable Auto-encoder

The main idea of this method is exchangeability, or permutation equivariance (PE). Suppose that there are a set of users $\mathbb{N} = \{1, \ldots, N\}$, a set of items $\mathbb{M} = \{1, \ldots, M\}$ as well as their interaction as a set of tuples $\mathbb{X} = \langle n, m, x \rangle$, where $n \in \mathbb{N}$, $m \in \mathbb{M}$ and $x \in \mathbb{R}$. The goal is to learn a function that maps from $N \times M$ to $\mathbb{R}$. This is a typical matrix completion problem. However, we are particularly interested in an exchangeable matrix X, such that, for any column-wise or row-wise permutation of X, the set of ratings remains the same.

Given $X \in \mathbb{R}^{N \times M}$, a fully connected layer $\sigma(Wvec(X))$ with $W \in \mathbb{R}^{NM \times NM}$ is called an exchangeable matrix layer if, for all permutation matrices $G^{(N)} \in \{0,1\}^{N \times N}$ and $G^{(M)} \in \{0,1\}^{M \times M}$, permutation of the rows and columns results in the same permutations of the output:

$$vec^{-1}(\sigma(Wvec(G^{(N)} X G^{(M)}))) = G^{(N)} vec^{-1}(\sigma(Wvec(X)))G^{(M)}, \tag{28}$$

where $vec(\cdot)$ is a vectorization operation and $vec^{-1}(\cdot)$ denotes the inverse of the vectorization, for example $vec^{-1}(vec(X)) = X$. Be that as it may, the weight matrices satisfying the above constraint is too restrictive and actually yielding the following form:

$$W_{(n,m),(n',m')} = \begin{cases} w_1 & n = n', m = m', \\ w_2 & n = n', m \neq m', \\ w_3 & n \neq n', m = m', \\ w_4 & n \neq n', m \neq m', \end{cases} \tag{29}$$

which results in four parameters, among which the first one connects to $X_{n,m}$, the second one connects to the same row n, the thrid one connects to the same column m and the fourth one is shared by all the other connections. The insight about the shared parameters leads to the following equivalent definition an exchangeable matrix layer. Given a strictly monotonic function $\sigma(\cdot)$, a neural layer $\sigma(Wvec(X))$ is an exchangeable matrix layer iff the elements of the parameter matrix W are tied together such that the resulting fully connected layer simplifies to:

$$Y = \sigma(w_1' X + \frac{w_2'}{N}(\mathbf{1}_N \mathbf{1}_N^T X) + \frac{w_3'}{M}(X \mathbf{1}_M \mathbf{1}_M^T) \tag{30}$$

$$+ \frac{w_4'}{NM}(\mathbf{1}_N \mathbf{1}_N^T X \mathbf{1}_M \mathbf{1}_M^T) + w_5' \mathbf{1}_N \mathbf{1}_M^T), \tag{31}$$

where $\mathbf{1}_N = [1, \ldots, 1]^T \in \mathbb{R}^{N \times 1}$ and $w_j' \in \mathbb{R}$.

For convolutional layers with multiple channels, the parameters are distinct for each combination of input and output channels. Given K input channels and O output channels, for a specific pair of channels $k \in K$ and $o \in O$, the corresponding scalar output is defined as:

$$Y_{n,m}^{\langle o \rangle} = \sigma(\sum_{k=1}^{K}(w_1^{\langle k,o \rangle} X_{n,m}^{\langle k \rangle} + \frac{w_2^{\langle k,o \rangle}}{N}(\sum_{n'} X_{n',m}^{\langle k \rangle}) \tag{32}$$

$$+ \frac{w_3^{\langle k,o \rangle}}{M}(\sum_{m'} X_{n,m'}^{\langle k \rangle}) + \frac{w_4^{\langle k,o \rangle}}{NM}(\sum_{n',m'} X_{n',m'}^{\langle k \rangle}) + w_5^{\langle o \rangle})). \tag{33}$$

Under the sparsity assumption, it is necessary to only take care of the non-zero entries. Assume that $\mathbb{R}_n = \{m | (n,m) \in \mathbb{I}\}$ is the set of non-zero elements in the n-th row and $\mathbb{C}_m$ denotes that in the m-th column, as expected, the terms in Equation 32 are transformed as follows:

$$\frac{w_2^{\langle k,o \rangle}}{N}(\sum_{n'} X_{n',m}^{\langle k \rangle}) \rightarrow \frac{w_2^{\langle k,o \rangle}}{|\mathbb{C}_m|}(\sum_{n' \in \mathbb{C}_m} X_{n',m}^{\langle k \rangle}), \tag{34}$$

$$\frac{w_3^{\langle k,o \rangle}}{M}(\sum_{m'} X_{n,m'}^{\langle k \rangle}) \rightarrow \frac{w_3^{\langle k,o \rangle}}{|\mathbb{R}_n|}(\sum_{m' \in \mathbb{R}_n} X_{n,m'}^{\langle k \rangle}), \tag{35}$$

$$\frac{w_4^{\langle k,o \rangle}}{NM}(\sum_{n',m'} X_{n',m'}^{\langle k \rangle}) \rightarrow \frac{w_4^{\langle k,o \rangle}}{|\mathbb{I}|}(\sum_{(n',m') \in \mathbb{I}} X_{n',m'}^{\langle k \rangle}), \tag{36}$$

which is adapted to the sparse settings.

In an inductive matrix completion setting, the rows and columns of the training matrices and test matrices share no overlapping entries, which is sometimes called matrix extrapolation. In contrast to the transductive setting, the indices are completely disjointed in the inductive setting. As an example, a system trained on a movie recommendation dataset can be applied to music recommendation.

The final architecture based on the above formulation is called Factorized Exchangeable Auto-encoder (FEA) [16]. Similar to GCMC, FEA consists of an encoder and a decoder. The encoder f_{enc} maps the input matrix $X \in \mathbb{R}^{N \times M \times K}$ to a row factor matrix $Z_N \in \mathbb{R}^{K_N \times N}$ and a column factor matrix $Z_M \in \mathbb{R}^{K_M \times M}$, in which the input passes through a composition of exchangeable matrix layers and pooling layers to be divided into Z_N and Z_M. The decoder g_{dec} reconstructs X' from Z_N and Z_M based on a composition of exchangeable matrix layers as well.

In optimization, the loss function is thus the classical one for encoder-decoder architectures, $\mathcal{L}(X, g_{dec}(f_{enc}(X)))$. A channel-level dropout approach is adopted for robustness. A conditional sampling strategy is reducing memory consumption to fit the data into the limited GPU memory in order to take advantage of the whole dataset. The sampling method first samples a subset of rows from the row marginal distribution and then selects a subset of columns from the sampled rows. The row and the conditional column distributions are as follows:

$$P(n) = \frac{|\mathbb{R}_n|}{\mathbb{I}}, \tag{37}$$

$$P(m|\mathbb{N}') = \frac{|\{(m,n) \in \mathbb{I} | n \in \mathbb{N}'\}|}{|\{(m',n) \in \mathbb{I} | n \in \mathbb{N}'\}|}, \tag{38}$$

where $\mathbb{N}' \subseteq \mathbb{N} = \{1, \ldots, N\}$ is a subset of rows.

4 EXPERIMENTS

In this section, we describe the adopted datasets, implementation details and the experiment results of the three implemented models on three datasets as well as some relevant analysis.

4.1 Datasets

We evaluated our three recommendation algorithms on a number of common collaborative filtering benchmark datasets: MovieLens 10M, Douban Movie, and Yahoo Music. The datasets consist of user ratings for items (such as movies and music) and incorporate additionally optional user/item features. A brief summary of the adopted datasets is as follows.

- MovieLens 1M dataset.
  Link - https://grouplens.org/datasets/movielens/1m/
  Size - 6 MB
- Douban Movie Short Comments Dataset V2.
  Link - https://www.kaggle.com/utmhikari/doubanmovieshortcomments
- Yahoo! Music user ratings of musical tracks, albums, artists and genres.
  Link - https://webscope.sandbox.yahoo.com/catalog.php?datatype=c
  Size - 1.5 GB

MovieLens 1M dataset [15] contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000. All ratings are contained in the file "ratings.dat" and are in the following format: UserID::MovieID::Rating::Timestamp. UserIDs range between 1 and 6,040. MovieIDs range between 1 and 3,952. Ratings are made on a 5-star scale (whole-star ratings only). Timestamp is represented in seconds since the epoch as returned by time. Each user has at least 20 ratings.

Douban Movie [21] is a Chinese website that allows Internet users to share their comments and viewpoints about movies. Users are able to post short or long comments on movies and give them marks. This dataset contains more than 2 million short comments of 28 movies in Douban Movie website. It can be used on text classification, text clustering, sentiment analysis, semantic web construction and some other fields that relate to web mining or NLP (of Chinese lol).

Yahoo! Music [8] offers a wealth of information and services related to many aspects of music. This dataset represents a snapshot of the Yahoo! Music community's preferences for various musical items. A distinctive feature of this dataset is that user ratings are given to entities of four different types: tracks, albums, artists, and genres. In addition, the items are tied together within a hierarchy. That is, for a track we know the identity of its album, performing artist and associated genres. Similarly we have artist and genre annotation for the albums. The dataset contains ratings provided by true Yahoo Music customers during 1999-2009. Both users and items (tracks, albums, artists and genres) are represented as meaningless anonymous numbers.

The Douban and Yahoo datasets follow the preprocessing steps in [21] while the MovieLens-1M dataset is split into a 20% test set and the remaining 80% being the training set, in which 10% is held out for validation.

4.2 Implementation Details

All the methods were implemented in Python and the TensorFlow library [1]. The experiments were conducted in the Google Colab environment [4]. The default parameter settings were adopted as in the original papers except that we run 1,000 epochs for each dataset and method for better comparisons because FEA generally takes a longer period of time to converge. Note that FMs are only able to make use of the rating matrices while the other methods take advantage of the side information such as the user profile feature vectors, the review text and item attributes.

4.3 Results

Table 1: The test loss for three datasets and three methods. The test loss evaluated at the minimum validation loss is reported for FEA while the final test loss values are given for the other two methods.

Method	Yahoo	Douban	ML-1M
FM	19.732	0.736	0.849
GCMC	19.797	0.753	0.849
FEA	20.430	0.736	0.892

The training and validation loss values for each method and each dataset for all the epochs are plotted in Figure 1 while the final test losses are shown in Table 1. The test losses after the final epoch are reported for FM and GCMC while the test loss obtained when the minimum evaluation loss is achieved is reported for FEA. All the loss values were evaluated in terms of the Root Mean Square Error (RMSE).

As shown in Figure 1, GCMC shows stable behavior in terms of losses and gradient descending. FEA is good at tackling large datasets such as Douban and ML-1M while it fails in the simpler Yahoo dataset. Furthermore, FEA exhibits the behavior that it suddenly finds a gradient direction leading to huge loss reduction in some early epochs. The fact the validation loss of FEA on the Yahoo dataset is increasing in the last hundreds of epoch might imply that the model is susceptible to overfitting because of its large hypothesis capacity. FMs generally have lower training losses than validation losses and converge quickly in the first 100 epochs. The strange trends in the Yahoo dataset for all the methods are likely to be as a result of the impurity in the dataset splitting step.

As shown in Table 1, the most classic method, FM, has the best performance among all the methods including the two deep learning approaches, GCMC and FEA. One explanation is that GCMC and FEA require a large number of optimization iterations to finally converge to a good optimum, as indicated in Figure 1. Moreover, the objective function in deep learning models is often highly non-convex. Thus being stuck in a local minimum is common in practice. It also takes a lot of efforts to fine-tune the hyper-parameters in neural networks to make it exhibit significantly excellent performance.

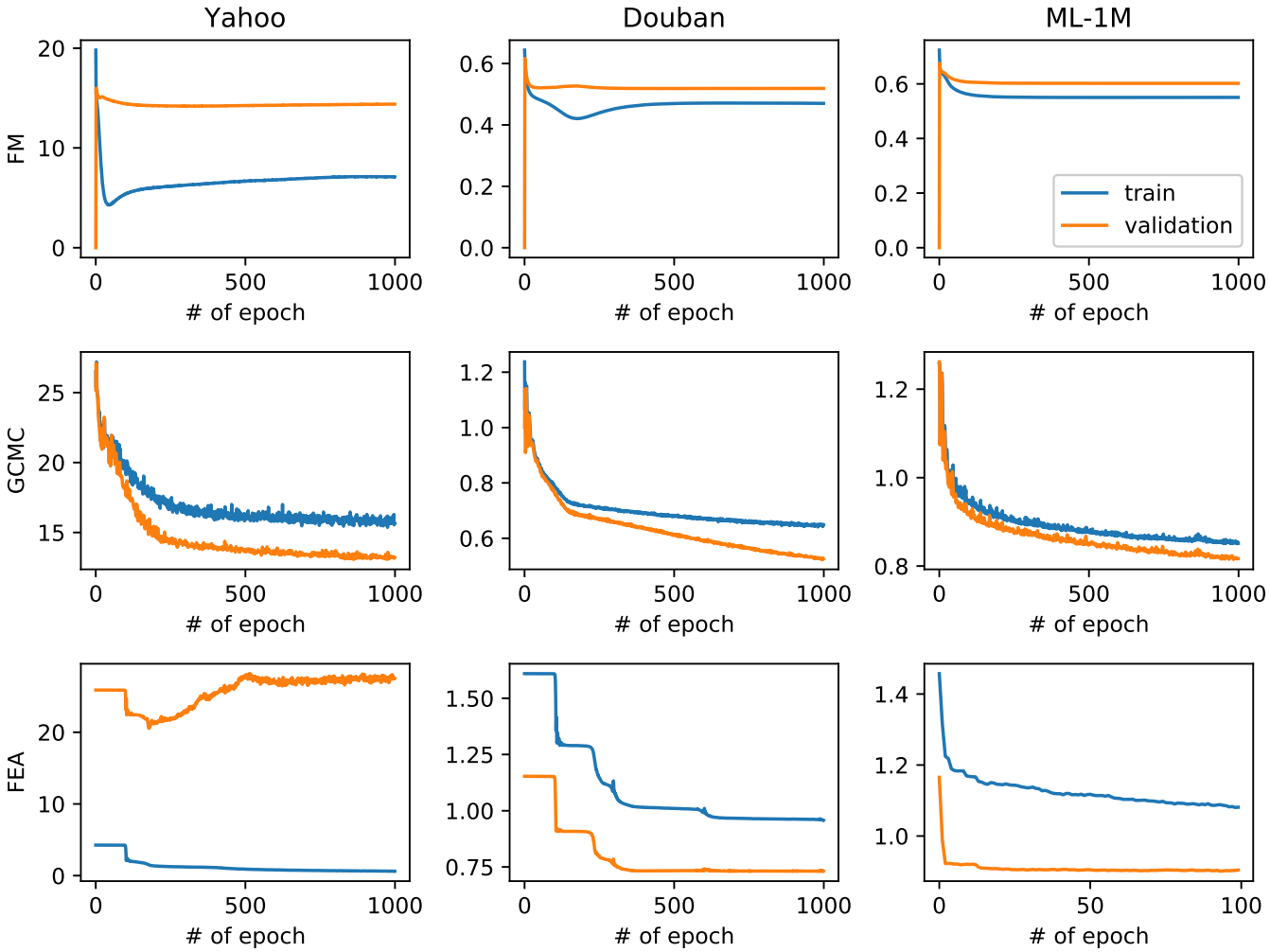

Figure 1: The training and validation epoch-wise loss values for three datasets and three models.

5 CONCLUSION AND FUTURE WORK

In this paper, we implemented three methods for recommender systems, including FM, GCMC and FEA. The methods were evaluated on three widely adopted publicly available datasets, Yahoo, Douban and ML-1M. Some experiment results were presented in the original papers while some results are unveiled. We made a detailed analysis on experiment results to explain some interesting exhibited behaviors.

Anticipating future work lies in applying the methods in other practical setting and more complicated datasets. FEA presents a promising research direction on transfer learning for recommender systems. Adversarial formulation including optimal transport [29] and Wasserstein distance [13] could be incorporated for a more robust and effective approach. Theoretical work on link prediction could be studied to find a way of applying the corresponding general methods of link prediction on general probabilistic graphical models to the recommender systems.

ACKNOWLEDGMENTS

The authors would like to thank Cornelia Caragea for the lecture.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263* (2017).

[3] Emmanuel J Candès and Benjamin Recht. 2009. Exact matrix completion via convex optimization. *Foundations of Computational mathematics* 9, 6 (2009), 717.

[4] Tiago Carneiro, Raul Victor Medeiros Da Nóbrega, Thiago Nepomuceno, Gui-Bin Bian, Victor Hugo C De Albuquerque, and Pedro Pedrosa Reboucas Filho. 2018. Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications. *IEEE Access* 6 (2018), 61677–61685.

[5] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*. 3844–3852.

[6] Ai-Lin Deng, Yang-Yong Zhu, and Bai-Le Shi. 2003. A collaborative filtering recommendation algorithm based on item rating prediction. *Journal of software* 14, 9 (2003), 1621–1628.

[7] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*. 1269–1277.

[8] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. 2011. The yahoo! music dataset and kdd-cup'11. In *Proceedings of the 2011 International Conference on KDD Cup 2011-Volume 18*. JMLR. org, 3–18.

[9] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*. 2224–2232.

[10] Gintare Karolina Dziugaite and Daniel M Roy. 2015. Neural network matrix factorization. *arXiv preprint arXiv:1511.06443* (2015).

[11] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2011. MyMediaLite: A free recommender system library. In *Proceedings of the fifth ACM conference on Recommender systems*. ACM, 305–308.

[12] Gustavo Gonzalez, Josep Lluís De La Rosa, Miquel Montaner, and Sonia Delfin. 2007. Embedding emotional context in recommender systems. In *2007 IEEE 23rd international conference on data engineering workshop*. IEEE, 845–852.

[13] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*. 5767–5777.

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[15] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2016), 19.

[16] Jason Hartford, Devon R Graham, Kevin Leyton-Brown, and Siamak Ravanbakhsh. 2018. Deep models of interactions across sets. *arXiv preprint arXiv:1803.02879* (2018).

[17] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[18] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.

[19] Sheng Li, Jaya Kawale, and Yun Fu. 2015. Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 811–820.

[20] Andriy Mnih and Ruslan R Salakhutdinov. 2008. Probabilistic matrix factorization. In *Advances in neural information processing systems*. 1257–1264.

[21] Federico Monti, Michael Bronstein, and Xavier Bresson. 2017. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*. 3697–3707.

[22] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International Conference on Data Mining*. IEEE, 995–1000.

[23] Steffen Rendle. 2012. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)* 3, 3 (2012), 57.

[24] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*. ACM, 791–798.

[25] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80.

[26] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 111–112.

[27] Si Si, Kai-Yang Chiang, Cho-Jui Hsieh, Nikhil Rao, and Inderjit S Dhillon. 2016. Goal-directed inductive matrix completion. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1165–1174.

[28] Yang Song, Ali Mamdouh Elkahky, and Xiaodong He. 2016. Multi-rate deep learning for temporal recommendation. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 909–912.

[29] Cédric Villani. 2008. *Optimal transport: old and new*. Vol. 338. Springer Science & Business Media.

[30] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 1235–1244.

[31] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 353–362.

[32] Yin Zheng, Bangsheng Tang, Wenkui Ding, and Hanning Zhou. 2016. A neural autoregressive approach to collaborative filtering. *arXiv preprint arXiv:1605.09477* (2016).