

FACTOR-REDUCED MATRIX FACTORIZATION WITH A NEW INITIALIZATION METHOD

Gábor Farkas, Patrik Karakai and Péter Magyar

(Budapest, Hungary)

Communicated by László Szili

(Received 6 May 2026; accepted 1 September 2026)

Abstract. Recommender systems have become an indispensable component of digital life. At the outset of the second quarter of the twenty-first century, one of the most effective approaches for their implementation is matrix factorization (MF), a learning algorithm whose input is a matrix generated by users through rating items with which they are familiar. MF is a latent factor method: although the algorithm does not explicitly observe user and item attributes during training, it can nevertheless predict the extent to which an unknown item is expected to satisfy a given customer's preferences. In real-world applications, additional information is often available, from which certain item properties can be inferred. Incorporating item attributes into MF is commonly achieved by setting the initial values of the item feature matrix accordingly. In this paper, we present a method that leverages item attributes to generate suitable initialization values for the user feature matrix, thereby improving the predictive accuracy and convergence speed of the learning algorithm.

1. Introduction

Modern recommender systems serve as critical infrastructure across many digital platforms, enabling users to get personalized item suggestions. Over the past decades, computer scientists have developed numerous methods for constructing effective recommender systems. The choice among these approaches

Key words and phrases: Rating-based recommender system, hybrid matrix factorization, autoencoder.

2010 Mathematics Subject Classification: 68T99.

<https://doi.org/10.71352/ac.59.010926>

is primarily determined by the type and amount of data available. In the present work, we focus on rating-based datasets, which are typically generated on online platforms as follows.

1.1. Recommender systems for rating-based datasets

Companies seeking to deploy effective recommender systems on their online platforms typically invite visitors to register as users and to rate items, such as products, with which they are already familiar. The information collected in this manner, namely the ratings provided by users, is referred to as *explicit feedback*. By contrast, *implicit feedback* denotes indirect behavioral information from which the system infers user preferences without requiring users to express them explicitly in the form of ratings; common examples include clicks, views, purchases, and content playback.

This data collection strategy is widely used in contemporary online services. Users are generally asked to assign scores on a predefined scale, most commonly from one to five or from one to ten. For example, in the MovieLens dataset Netflix used a five-point scale, whereas IMDb employs a ten-point scale. Unless stated otherwise, throughout this work the term rating refers to a value measured on a five-point scale.

Through this process, the company constructs a database containing ratings assigned by registered users to selected items. In practice, however, such databases are highly incomplete, since an individual user is generally familiar with only a small proportion of the available items and may not necessarily be willing to rate every item they have encountered.

The objective is to develop a recommender system for given sets of users U and items I . More specifically, the task is to construct a model capable of predicting the rating that an arbitrary user would assign to any particular item, including items with which the user has had no prior experience. Based on these predictions, the system recommends to each user those items for which the estimated ratings are relatively high.

The problem can be modeled by a so-called *user-item rating matrix* $R_{n \times m}$, where n denotes the number of users and m denotes the number of items. The entry $r_{u,i}$ of the matrix R represents the rating assigned by user u to item i . Missing entries in R are treated as unobserved values (denoted as NULL). Due to the incompleteness of the collected data, R is a highly sparse matrix, meaning that it contains a large number of NULL values. Therefore, the central objective is to estimate these missing values.

1.2. Basics of matrix factorization

It can be observed that virtually no information is available about the users themselves; only certain historical user-item interactions are known. Based on these observations, *Collaborative Filtering* (CF) methods can be employed to predict a given user's preference for a previously unseen item. CF techniques

can achieve remarkably accurate results, provided that users' preferences do not change abruptly over time.

Latent factor methods (LFM) constitute an important class of collaborative filtering. *Matrix factorization* (MF) is one of the most widely used techniques for implementing latent factor models in recommender systems [7]. These methods aim to map users and items into a k -dimensional *latent space* on the basis of the interaction values observed between them. Consequently, the position of each user and item is represented by k coordinates, whose specific values are learned during a training process.

The central idea is that the interaction values should also be recoverable from this k -dimensional representation. More precisely, there should exist a mapping, referred to as the *rating prediction function*, that, given the positions of an arbitrary user u and item i , predicts the corresponding interaction value $r_{u,i}$. The training process continues until the coordinates of the users and items have been adjusted such that the rating prediction function provides an accurate approximation of the known $r_{u,i}$ values. Once this condition is achieved, the model is expected to generalize beyond the observed interactions and to provide accurate estimates of previously unknown values as well.

The term *latent factor method* is entirely appropriate, since user preferences and item characteristics are not directly observable but are instead represented by hidden factors learned from the interaction data.

Consider now the following approach: users and items are mapped into the latent space by means of k -dimensional *user feature* and *item feature vectors*, while the rating prediction function is defined as the dot product of these vectors. Thus, for n users and m items, matrix factorization produces the *user feature matrix* $P_{n \times k}$ and the *item feature matrix* $Q_{m \times k}$. The so-called *predicted user-item rating matrix* obtained from their product,

$$\hat{R}_{n \times m} = PQ^T$$

provides an approximation of R . More precisely,

$$\hat{r}_{u,i} \approx r_{u,i}, \text{ if } (u,i) \in \mathcal{K},$$

where

$$\mathcal{K} = \{(u,i) \mid r_{u,i} \neq \text{NULL}\}.$$

The remaining entries of $\hat{R}$ serve as predictions for the missing entries of R .

Although this formulation has proven effective, in more advanced matrix factorization models the dot product constitutes only one component of the overall rating approximation.

1.3. Evolution of matrix factorization

Researchers were naturally led to the idea of employing singular value decomposition (SVD) to approximate the missing entries of the user-item rating

matrix R . An early form of SVD was first described by E. Beltrami in 1873 [1]. Naturally, however, his work did not apply the method to matrix factorization. The next fundamental milestone was the Eckart–Young theorem, published by C. Eckart and G. Young in 1936 [4]. Their result provides an essential theoretical foundation for modern low-rank matrix approximation and for numerous matrix factorization methods.

One of the earliest influential applications of SVD in the field of collaborative filtering was introduced by D. Billsus and M. J. Pazzani in 1998 [2]. Their method applied singular value decomposition to a sparse matrix of user ratings in order to identify latent structure and to project users and items into a lower-dimensional space. The resulting representation was subsequently used as input to a machine learning model for predicting user preferences.

In 2000, Sarwar et al. investigated the application of dimensionality reduction in recommender systems and demonstrated how SVD could be used to address the scalability and sparsity problems associated with large user–item rating matrices [9].

J. F. Canny later proposed a probabilistic factor-analysis model for collaborative filtering, showing that latent factor representations could also be incorporated into privacy-preserving recommender methods [3].

A particularly important practical development occurred during the Netflix Prize competition. In 2006, Simon Funk published an online description of a gradient-descent-based latent factor model for predicting missing ratings [5]. The method directly learned low-dimensional user and item vectors from the observed entries of the rating matrix. Funk’s approach became widely known as *Funk-SVD*, although it does not compute the classical singular value decomposition of the complete rating matrix. Instead, it optimizes a regularized low-rank factor model using only the available ratings. This practical approach contributed substantially to the widespread adoption of matrix-factorization techniques in recommender systems.

1.4. SVD++

Building on Funk–SVD, Koren et al. introduced SVD++ to improve predictive accuracy by incorporating both explicit ratings and implicit behavioral signals [7].

The implicit feedback matrix $F \in \{0,1\}^{n \times m}$ is constructed as a binary projection of R

$$f_{u,i} = \begin{cases} 1 & \text{if } (u,i) \in \mathcal{K}, \\ 0 & \text{otherwise.} \end{cases}$$

The set of items rated by user u is defined as $N_u = \{i \in I \mid f_{u,i} = 1\}$.

To incorporate implicit behavioral signals, SVD++ introduces the implicit feedback item matrix $Y \in \mathbb{R}^{m \times k}$. While the standard item matrix Q captures characteristic profiles derived from rating values, each row $y_j \in \mathbb{R}^k$ in Y repre-

sents an alternative latent profile for item j that depends strictly on whether a user has interacted with an item or not ($f_{u,i} = 1$) rather than the score itself.

Furthermore, ratings often show systematic variations unrelated to specific user-item preferences, such as certain users giving consistently lower ratings or popular items receiving higher baseline scores. To account for these baseline deviations, SVD++ incorporates baseline bias terms [7]. The total predicted rating $\hat{r}_{u,i}$ for an observed user-item pair (u, i) is formulated as:

$$\hat{r}_{u,i} = \mu + b_u + b_i + q_i^T \left(p_u + \frac{1}{\sqrt{|N_u|}} \sum_{j \in N_u} y_j \right),$$

where

$$\mu = \frac{1}{|\mathcal{K}|} \sum_{(u,i) \in \mathcal{K}} r_{u,i}$$

represents the global mean rating across all training data, $b_u \in \mathbb{R}$ is the user bias, $b_i \in \mathbb{R}$ is the item bias, and $\frac{1}{\sqrt{|N_u|}}$ acts as a normalization factor ensuring that users with many ratings do not dominate the implicit factor sum [7].

The parameters $(b_u, b_i, p_u, q_i, y_j)$ are updated by minimizing the regularized squared error using Stochastic Gradient Descent (SGD) [7]:

$$\min_{P, Q, Y, b} \sum_{(u,i) \in \mathcal{K}} \left(e_{u,i}^2 + \lambda \left(\|p_u\|^2 + \|q_i\|^2 + \sum_{j \in N_u} \|y_j\|^2 \right) + \lambda_b (b_u^2 + b_i^2) \right),$$

where $e_{u,i} = r_{u,i} - \hat{r}_{u,i}$ denotes the prediction error, while λ and λ_b are the regularization parameters for the latent factors and bias terms, respectively. Parameter updates proceed iteratively via SGD:

$$b_u \leftarrow b_u + \alpha_b (e_{u,i} - \lambda_b b_u),$$

$$b_i \leftarrow b_i + \alpha_b (e_{u,i} - \lambda_b b_i),$$

$$p_u \leftarrow p_u + \alpha (e_{u,i} q_i - \lambda p_u),$$

$$q_i \leftarrow q_i + \alpha \left(e_{u,i} \left(p_u + \frac{1}{\sqrt{|N_u|}} \sum_{j \in N_u} y_j \right) - \lambda q_i \right),$$

where α and α_b denote the learning rates for latent factors and bias terms respectively. For each interacted item $j \in N_u$, the implicit vector is updated as:

$$y_j \leftarrow y_j + \alpha \left(e_{u,i} \frac{1}{\sqrt{|N_u|}} q_i - \lambda y_j \right).$$

Remark 1.1 (Terminological clarification). It is important to emphasize that referring to the vectors y_j as *implicit feature vectors* does not imply that the vectors themselves constitute directly observed implicit feedback. The implicit feedback is in fact encoded by the set N_u , which contains the items with which user u has previously interacted. By contrast, each vector y_j is a latent parameter vector associated with item j , whose values are learned during model training and which represents the contribution of the corresponding interaction to the user’s preference representation. Consequently, the set N_u determines which y_j vectors are included in the representation of a given user, whereas the components of these vectors are learned by minimizing the prediction error associated with the explicit ratings.

2. A new approach

Initially, the primary objective of matrix factorization was low-rank matrix approximation, namely, the decomposition of a high-dimensional and extremely sparse user–item interaction matrix into two lower-dimensional dense matrices representing users and items, respectively. Despite their effectiveness, classical matrix factorization algorithms tend to exhibit degraded performance when applied to highly sparse datasets [7]. Furthermore, the conventional strategy of randomly initializing the latent matrices often results in suboptimal local minima and slow convergence [8].

To address these limitations, we introduce the Factor-Reduced Matrix Factorization (FRMF) method. Our algorithm improves the convergence of low-rank representations by initializing the user feature matrix with values derived from the items. This approach reflects real-world scenarios, in which extensive information is often available about certain items, whereas comparatively little is known about the users.

In addition, FRMF accelerates the model-training process by reducing the computational complexity associated with a high-dimensional latent space. Our method extends the work of Zhao et al., who utilized the item attribute information available in the dataset to initialize the item feature matrix [11]. We further extended their approach by introducing an autoencoder-based method that is also capable of effectively initializing the user latent factors. The architecture of FRMF is illustrated in Figure 1.

The specific dataset used in our experiments was MovieLens 100K [6]. This dataset was selected because it is one of the most widely used benchmarks for matrix factorization experiments and because it enables a direct comparison of our results with those reported by Zhao et al. [11].

The dataset contains 100,000 ratings provided by 943 users for 1,682 items. Ratings are assigned on a scale ranging from 1 to 5. In addition, the dataset provides 18 binary genre indicators for each item. Multiple genre labels may be associated with a single item.

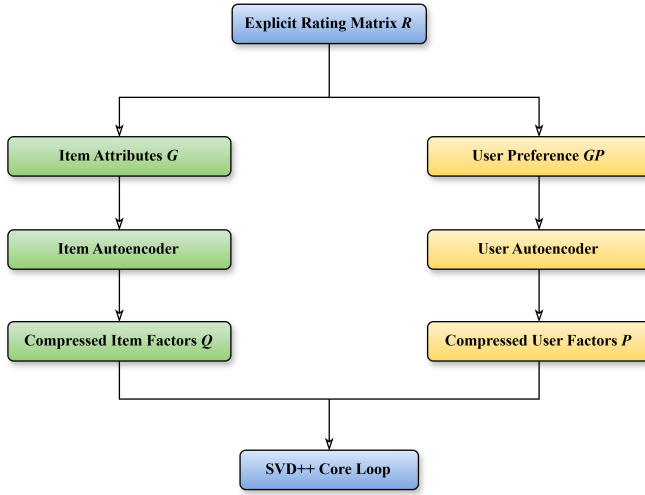


Figure 1. Architectural overview of FRMF

2.1. Autoencoders in the model

Standard PCA has difficulty capturing the complex relationships between item attributes and user preferences. Consequently, autoencoders have been widely adopted in recent years for the initialization of item feature vectors [10].

An autoencoder is a neural network that encodes the input data into a latent representation and subsequently reconstructs the original input from this representation. During training, the discrepancy between the input and its reconstruction is minimized. Its objective is to automatically identify the essential features of the data and construct a compact representation that can be used, among other purposes, for dimensionality reduction and feature extraction. In the fully connected architectures considered in this paper, the number of neurons in a layer is equal to the dimensionality of the representation produced by that layer.

Remark 2.1 (Important assumption). Throughout this paper, we consider only autoencoders in which the latent representation has a lower dimensionality than the input layer; such models are known as *undercomplete autoencoders*.

Each autoencoder employed in this study includes exactly one input layer and one output layer, together with one or more hidden layers. Among the hidden layers, a unique layer can be identified, known as the *bottleneck layer* or *code layer*. It is the layer with the smallest dimensionality in the network and produces the compressed latent representation of the input. Accordingly, the network includes an encoder block, whose input is the input layer, and a decoder block, whose output is the output layer. The bottleneck layer simultaneously serves as the output of the encoder and the input of the decoder.

We illustrate the architecture of the autoencoders employed in this study by means of an example. Let the dimensionality of the input data be $k = 18$, in accordance with the MovieLens dataset, let the dimensionality of the bottleneck layer be $d = 12$, and let $l = 3$ denote the number of hidden layers in each of the encoder and decoder blocks. The numbers of neurons in the intermediate layers are calculated using the linspace function from NumPy package.

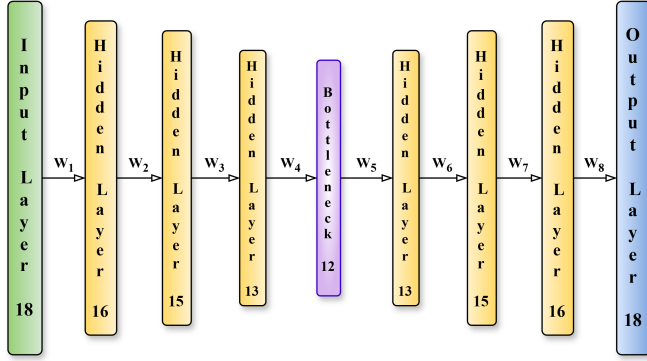


Figure 2. Layers and dimensions in the autoencoder.

Let $\mathbf{W}_i$ denote the trainable weight matrix associated with the i -th transformation of the autoencoder. Each matrix maps the representation produced by one layer to the vector space corresponding to the subsequent layer. Assuming that the layer representations are column vectors, a layer transformation is defined as $\mathbf{z}_i = \sigma_i(\mathbf{W}_i \mathbf{z}_{i-1} + \mathbf{b}_i)$, where b_i is the i -th bias vector and σ_i is the i -th activation function. The dimensions of the encoder weight matrices are

$$\mathbf{W}_1 \in \mathbb{R}^{16 \times 18}, \mathbf{W}_2 \in \mathbb{R}^{15 \times 16}, \mathbf{W}_3 \in \mathbb{R}^{13 \times 15} \text{ and } \mathbf{W}_4 \in \mathbb{R}^{12 \times 13}.$$

Similarly, the decoder weight matrices satisfy

$$\mathbf{W}_5 \in \mathbb{R}^{13 \times 12}, \mathbf{W}_6 \in \mathbb{R}^{15 \times 13}, \mathbf{W}_7 \in \mathbb{R}^{16 \times 15} \text{ and } \mathbf{W}_8 \in \mathbb{R}^{18 \times 16}.$$

The architecture of the autoencoder is illustrated in Figure 2. The matrices $\mathbf{W}_1, \dots, \mathbf{W}_4$ progressively compress the input into the d -dimensional latent representation, whereas $\mathbf{W}_5, \dots, \mathbf{W}_8$ transform this representation back into the original 18-dimensional input space. The bias vectors $\mathbf{b}_i$ and the activation functions σ_i are omitted from the arrows for notational simplicity.

The model is trained for 300 epochs using the Adam optimizer implemented in PyTorch, with the Mean Squared Error (MSE) serving as the loss function. Since our project initializes both the item latent factor matrix Q and the user latent factor matrix P , we employ two separate autoencoders, referred to as the *item autoencoder* (I_{encoder}) and the *user autoencoder* (U_{encoder}). The *item attribute matrix* G and the *user genre preference matrix* GP are provided as

inputs to the input layers of the I_{encoder} and the U_{encoder} , respectively. The construction of these matrices is discussed below. Upon completion of the autoencoder training process, the dimension-reduced representations are obtained from the corresponding code layers. The values contained in these representations are subsequently rescaled to the interval $[0, 1]$ using min-max normalization and are then used to initialize Q and P , respectively.

2.2. Item attribute matrix

The contribution reported by Zhao et al. [11] was the combination of attribute mapping-based and neural network-based initialization techniques. In the proposed FRMF model, the item latent factors are initialized using their method and the predefined genre attributes provided in the MovieLens dataset.

Let $G \in \{0, 1\}^{m \times k}$ represent the item attribute matrix, where k is the number of genres (in this study $k = 18$). For any item i , its attribute vector $G_i = [g_{i,1}, g_{i,2}, \dots, g_{i,k}]$ is defined as:

$$g_{i,a} = \begin{cases} 1 & \text{if item } i \text{ possesses attribute } a, \\ 0 & \text{otherwise.} \end{cases}$$

$I_{\text{encoder}}(G)$ denotes the result obtained by applying the item autoencoder to the input G .

2.3. User genre preference matrix

While the item attributes are available in the dataset, user preferences, as in real-world settings, cannot be observed directly. To bridge this gap, we construct a *user genre preference matrix*, denoted by $GP \in \mathbb{R}^{n \times k}$.

Consider an arbitrary user u who has interacted with s items ($s \leq m$), that is, ($s = |N_u|$). Let $R^{(u)} \in \mathbb{R}^{1 \times s}$ denote the row vector containing the historical numerical ratings provided by user u . Furthermore, let $G^{(u)} \in \{0, 1\}^{s \times k}$ denote the submatrix of G that contains the attribute vectors of the items belonging to N_u .

The item-specific ratings provided by user u are transformed into attribute-wise aggregated scores through the following matrix multiplication:

$$X^{(u)} = R^{(u)} \cdot G^{(u)},$$

where $X^{(u)} \in \mathbb{R}^{1 \times k}$. The t -th component of the vector $X^{(u)}$ is given by

$$X_t^{(u)} = \sum_{i \in N_u} r_{u,i} g_{i,t}.$$

To normalize these values, we divide by the maximum theoretical aggregate rating that a user could assign across all rated items. Since the maximum rating

in the MovieLens 100K dataset is 5, we define

$$GP^{(u)} = \frac{R^{(u)} \cdot G^{(u)}}{5s}$$

where $GP^{(u)} \in (0, 1]^k$ represents the normalized preference vector for user u . Concatenating these vectors for all n users gives us the complete user genre preference matrix $GP \in \mathbb{R}^{n \times k}$.

3. Factor reduced matrix factorization algorithm

The core of FRMF is based on the SVD++ model proposed by Koren et al. [7] that we described in Section 1.4.

While standard SVD++ initializes the latent factor matrices P and Q with random values drawn from a zero-mean Gaussian distribution with a small standard deviation [8], Factor-Reduced Matrix Factorization (FRMF) departs substantially from this conventional practice. The matrix Q is initialized using the hybrid method proposed by Zhao et al. [11], whereas P is initialized using our own method, which is described in the following.

The bias parameters b_u , b_i and implicit feedback factors Y are initialized by the Gaussian distribution $\mathcal{N}(0, 0.1^2)$. The notation $b_u[u]$ and $b_i[i]$ means the vector of b_u belongs to user u and the vector of b_i belongs to item i respectively. Parameter updates are optimized using the standard SVD++ SGD formulations defined in Section 1.4.

Among the parameters of the algorithm, the matrices $R \in \mathbb{R}^{n \times m}$, F , G , and GP have already been discussed, and the data contained in these matrices are globally accessible. The global mean rating is denoted by μ . The latent dimensionality is set to $d = 12$, the number of epochs to $T = 60$, the learning rate to $\alpha = 0.01$, and the regularization parameter to $\lambda = 0.1$. For the bias terms, the learning rate and regularization parameter are set to $\alpha_b = 0.01$ and $\lambda_b = 0.01$, respectively.

Function FRMF($\mu, d, T, \alpha, \lambda, \alpha_b, \lambda_b$)

```

1: global  $R, F, G, GP$ 
2:  $b_u \leftarrow \mathcal{N}(0, 0.1^2)$ 
3:  $b_i \leftarrow \mathcal{N}(0, 0.1^2)$ 
4:  $P \leftarrow U_{\text{encoder}}(GP)$ 
5:  $Q \leftarrow I_{\text{encoder}}(G)$ 
6:  $Y \leftarrow \mathcal{N}(0, 0.1^2)$ 
7: for  $t = 1, \dots, T$  do
8:   for  $u = 1, \dots, n$  do
9:      $\text{implicit\_sum} \leftarrow \frac{1}{\sqrt{|N_u|}} \cdot \sum_{j \in N_u} y_j$ 
10:    for  $i \in N_u$  do
```

```

11:          $\hat{r}_{u,i} \leftarrow \mu + b_u[u] + b_i[i] + Q[i] \cdot (P[u] + \text{implicit\_sum})$ 
12:          $e_{u,i} \leftarrow r_{u,i} - \hat{r}_{u,i}$ 
13:          $b_u[u] \leftarrow b_u[u] + \alpha_b(e_{u,i} - \lambda_b \cdot b_u[u])$ 
14:          $b_i[i] \leftarrow b_i[i] + \alpha_b(e_{u,i} - \lambda_b \cdot b_i[i])$ 
15:          $P[u] \leftarrow P[u] + \alpha(e_{u,i} \cdot Q[i] - \lambda \cdot P[u])$ 
16:          $Q[i] \leftarrow Q[i] + \alpha(e_{u,i} \cdot (P[u] + \text{implicit\_sum}) - \lambda \cdot Q[i])$ 
17:          $Y[N_u] \leftarrow Y[N_u] + \alpha(e_{u,i} \cdot \frac{1}{\sqrt{|N_u|}} \cdot Q[i] - \lambda \cdot Y[N_u])$ 
18:     end for
19: end for
20:     Compute training loss (MSE + Regularization terms)
21: end for
22: return  $P, Q, Y, b_u, b_i$ 

```

Algorithm 1: Pseudocode of Factor-Reduced Matrix Factorization

The algorithm iterates over all users in the training set, computes the implicit feedback sum for each user, and subsequently iterates over the item indices. It then checks the actual rating assigned by the user to a given item and, if this value is not NULL, computes the corresponding predicted rating. If the movie belongs to N_u , additional information about the user's interaction with that movie is available, thereby enabling a more accurate prediction. If the movie does not belong to N_u , only an approximate rating can be obtained on the basis of the global mean and the user bias.

4. Results

To quantify the accuracy of our predictions, two evaluation metrics are implemented: Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). Both metrics assess the difference between the actual ratings and the predicted ratings over the test set, the difference between them being that RMSE penalizes larger errors more than MAE.

During the training process the model optimization is based on RMSE values, but MAE is also reported as an additional evaluation metric.

The objective is to minimize the RMSE over the set of observed ratings where $f_{u,i} = 1$. For any observed user-item pair, the prediction error $e_{u,i}$ is calculated as:

$$e_{u,i} = r_{u,i} - \hat{r}_{u,i}.$$

4.1. Autoencoder

Autoencoder training was performed on an RTX 3070 GPU. The loss function is visualized in Figure 3. It follows a typical decreasing loss curve, which means the autoencoder is able to compress the data effectively to lower dimensions.

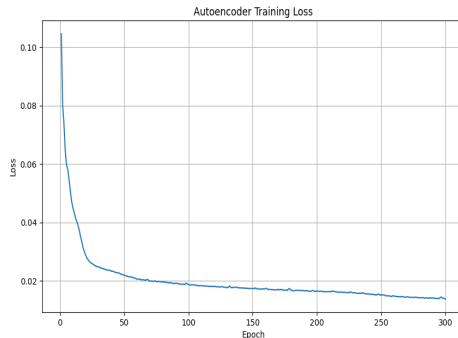


Figure 3. Autoencoder training loss

4.2. Performance of different initialization methods

We have compared seven distinct configurations of the SVD++ architecture, as summarized in Table 1. The Item Attribute Init and Compressed Item Init were both used by Zhao et al. in their paper, so we can directly compare our results to Zhao et al. [11].

Name	Initialized	Method	d
Random Init	none	$\mathcal{N}(0, 0.1^2)$	18
Item Attribute Init	items	attribute mapping	18
Compressed Item Init	items	compressed attribute mapping	12
User Attribute Init	users	attribute mapping	18
Compressed User Init	users	compressed attribute mapping	12
Both Attributes Init	items & users	attribute mapping	18
Both Compressed Init	items & users	compressed attribute mapping	12

Table 1. The seven different initialization methods

Both Attributes Init achieved the best improvement over Random initialization indicating that initializing both items and users improves accuracy. The performance gains achieved by these initializations are shown in Figure 4.

Compressing the factors by 33% preserves most of the accuracy. The Both Compressed Init model still has a 0.5% RMSE improvement over the uncompressed Random Init baseline, while using a smaller, more efficient latent model.

4.3. Cross-validation

To verify that the performance gains are not overfitted to a specific train-test split, we perform a rigorous 5-fold cross-validation. The data is split into 5 disjoint subsets, iteratively utilizing four folds for parameter training and the remaining fold for validation. The results of the cross-validation result and their averages are shown in Figures 5. and 6.

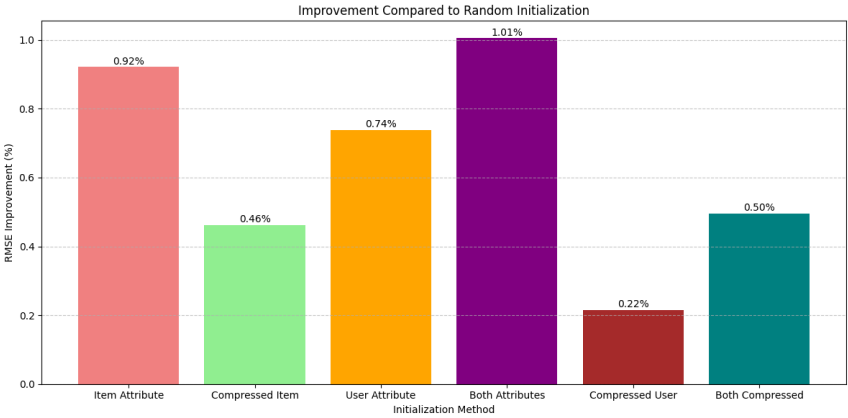


Figure 4. Relative RMSE improvement over Random Init

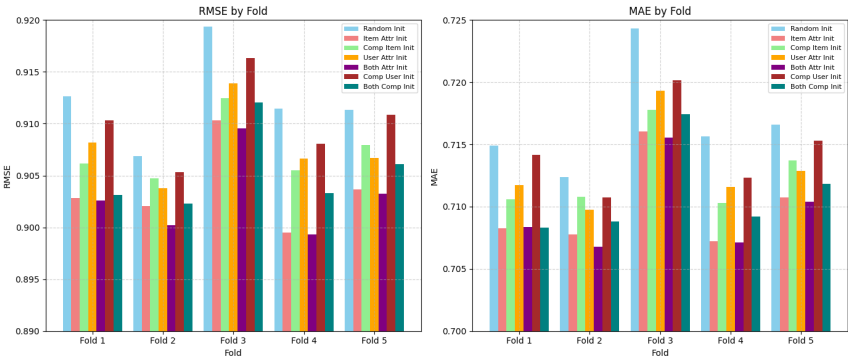


Figure 5. Cross-validation results across folds

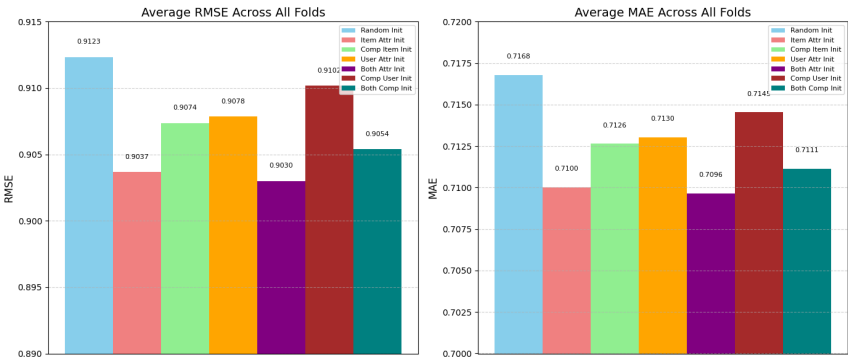


Figure 6. Cross-validation average

The cross-validation results confirm our previous findings. The Both Attributes Init model consistently achieves the lowest prediction error across all folds. The limited variation in its performance across the folds further indicates the robustness and stability of the proposed initialization technique.

4.4. Dimensionality reduction analysis

To determine the optimal boundary for compression, we trained a series of autoencoders with output dimensions ranging from $d = 18$ to $d = 6$. For each tested target dimension d , the intermediate layer dimensions were dynamically calculated using the same linspace function described in Section 2.1 with $l = 3$ hidden layers. Figure 7. illustrates the RMSE curves mapped against the uncompressed baselines.

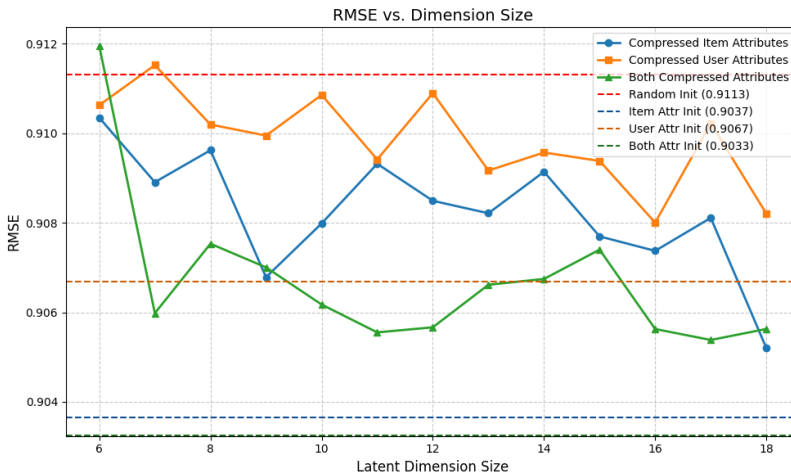


Figure 7. RMSE over different latent dimension sizes compared to baselines

With a 66.7% reduction in latent dimensionality, two out of three models outperform the Random Init baseline. This highlights that neural compression retains the core features, while still filtering out noise when compressing the data. Furthermore, factor reduction translates directly into computational runtime speedups during collaborative filtering. Because SVD++ performs gradient updates across d dimensions for every rated interaction, reducing $d = 18$ down to $d = 12$ reduces per-epoch floating-point operations by 33.3%. Training 60 epochs of a baseline 18-factor SVD++ on the MovieLens 100K dataset required 3.65s, whereas 12-factor FRMF completed in 2.47s which is a 32.3% speedup.

4.5. Evaluation on large-scale datasets: MovieLens 1M and 10M

To address the scalability of the proposed method and determine whether the performance advantages of dual initialization and factor reduction hold

on large-scale, sparse real-world systems, we extended our evaluation to the MovieLens 1M and MovieLens 10M datasets.

Table 2. presents the 5-fold cross-validation results across all three dataset scales alongside the SVD++ matrix factorization execution runtimes. Here, the SVD++ time denotes Both Attributes Init’s training time, and FRMF Time denotes Both Compressed Init’s training time.

Initialization Method	MovieLens 100K		MovieLens 1M		MovieLens 10M	
	RMSE	MAE	RMSE	MAE	RMSE	MAE
Random Init	0.9123	0.7167	0.8691	0.6849	0.8266	0.6375
Item Attribute Init	0.9037	0.7100	0.8667	0.6829	0.8237	0.6353
Compressed Item Init	0.9063	0.7121	0.8696	0.6851	0.8256	0.6368
User Attribute Init	0.9078	0.7130	0.8680	0.6839	0.8235	0.6351
Compressed User Init	0.9094	0.7145	0.8694	0.6850	0.8244	0.6358
Both Attributes Init	0.9030	0.7097	0.8662	0.6825	0.8224	0.6343
Both Compressed Init	0.9054	0.7109	0.8678	0.6838	0.8237	0.6353
SVD++ Time	3.65 s		55.46 s		592.29 s	
FRMF Time ($d = 12$)	2.47 s		39.75 s		445.91 s	
Runtime Reduction	32.3%		28.3%		24.7%	

Table 2. 5-fold cross-validation performance (RMSE, MAE) and SVD++ training runtimes across MovieLens 100K, 1M, and 10M datasets over 60 epochs.

Across all three datasets Both Attributes Init consistently achieves the lowest error prediction, confirming that initializing both users and item latent matrices is always beneficial.

In every tested dataset, Both Compressed Init outperformed the standard Random Init baseline. This confirms that autoencoder-based compression successfully retains the crucial information from the original data.

References

- [1] **Beltrami, E.**, Sulle funzioni bilineari, *Giornale di Matematiche ad Uso degli Studenti delle Università*, **11** (1873), 98–106.
<https://www-users.cse.umn.edu/~boley/publications/papers/Beltrami.pdf>
- [2] **Billsus, D. and M. J. Pazzani**, Learning Collaborative Information Filters, *Proceedings of the Fifteenth International Conference on Machine Learning*, Morgan Kaufmann, 1998, 46–54.
<https://www.ics.uci.edu/~pazzani/Publications/MLC98.pdf>
- [3] **Canny, J. F.**, Collaborative Filtering with Privacy via Factor Analysis, *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, 2002, 238–245.
<https://doi.org/10.1145/564376.564419>

- [4] **Eckart, C. and G. Young**, The Approximation of One Matrix by Another of Lower Rank, *Psychometrika*, **1** (1936), 211–218.
<https://doi.org/10.1007/BF02288367>
- [5] **Funk, S.**, Try This at Home, *Simon Funk's Weblog* (2006).
<https://sifter.org/~simon/journal/20061211.html>
- [6] **Harper, F. M. and J. A. Konstan**, The MovieLens datasets: History and context, *ACM Transactions on Interactive Intelligent Systems (TiiS)*, **5** (4) (2015), 1–19.
<https://doi.org/10.1145/2827872>
- [7] **Koren, Y., R. Bell and C. Volinsky**, Matrix factorization techniques for recommender systems, *Computer*, **42** (8) (2009), 30–37.
<https://doi.org/10.1109/MC.2009.263>
- [8] **Rendle, S. and L. Schmidt-Thieme**, Online-updating regularized kernel matrix factorization models for large-scale recommender systems, *Proceedings of the 2008 ACM Conference on Recommender Systems (RecSys '08)*, (2008), 251–258.
<https://doi.org/10.1145/1454008.1454047>
- [9] **Sarwar, B. M., G. Karypis, J. A. Konstan and J. T. Riedl**, Application of Dimensionality Reduction in Recommender System: A Case Study, *University of Minnesota, Department of Computer Science and Engineering, Technical Report 00-043*, 2000.
<https://hdl.handle.net/11299/215429>
- [10] **Sedhain, S., A. K. Menon, S. Sanner and L. Xie**, Autorec: Autoencoders meet collaborative filtering, *Proceedings of the 24th International Conference on World Wide Web*, (2015), 111–112.
<https://doi.org/10.1145/2740908.2742726>
- [11] **Zhao, J., X. Geng, J. Zhou, Q. Sun, Y. Xiao, Z. Zhang and Z. Fu**, Attribute mapping and autoencoder neural network based matrix factorization initialization for recommendation systems, *Knowledge-Based Systems*, **166** (2019), 132–139.
<https://doi.org/10.1016/j.knosys.2018.12.022>

Gábor Farkas

<https://orcid.org/0000-0003-0629-8699>

Eötvös Loránd University

Department of Computer Algebra

Budapest

Hungary

farkasg@inf.elte.hu

Patrik Karakai

<https://orcid.org/0009-0000-9720-3274>

Eötvös Loránd University

Department of Computer Algebra

Budapest

Hungary

gi8v0o@inf.elte.hu

Péter Magyar

<https://orcid.org/0000-0002-1581-1143>

Eötvös Loránd University

Department of Computer Algebra

Budapest

Hungary

z47mli@inf.elte.hu

