# Movie Recommendations Using the Deep Learning Approach

Jeffrey Lund
*Computer Science Department*
*Brigham Young University*
*Provo, Utah 84602, USA*
*jefflund@byu.edu*

Yiu-Kai Ng
*Computer Science Department*
*Brigham Young University*
*Provo, Utah 84602, USA*
*ng@compsci.byu.edu*

***Abstract*—Recommendation systems are an important part of suggesting items especially in streaming services. For streaming movie services like Netflix, recommendation systems are essential for helping users find new movies to enjoy. In this paper, we propose a deep learning approach based on autoencoders to produce a collaborative filtering system which predicts movie ratings for a user based on a large database of ratings from other users. Using the MovieLens dataset, we explore the use of deep learning to predict users' ratings on new movies, thereby enabling movie recommendations. To verify the novelty and accuracy of our deep learning approach, we compare our approach to standard collaborative filtering techniques: k-nearest-neighbor and matrix-factorization. The experimental results show that our recommendation system outperforms a user-based neighborhood baseline both in terms of root mean squared error on predicted ratings and in a survey in which users judge between recommendations from both systems.**

***Keywords*-movie recommendation; deep learning; collaborative filtering**

## I. Introduction

Movie streaming services like Netflix, Hulu, Amazon Prime, and others are increasingly used by consumers to enjoy video content. For example, in 2017 Netflix subscribers collectively watched more than 140 million hours per day[1] and Netflix surpassed $11 billion in revenue in 2017.[2] In fact, roughly 80% of hours streamed at Netflix were influenced by their proprietary recommendation system [5]. Undoubtedly, movie streaming services have become an integral part of how we consume video content today, and the importance of movie recommendation systems cannot be understated—they are an integral part of how we consume video content today. With this in mind, the problem we propose to work on is movie recommendations through collaborative filtering based on the deep learning strategy.

For movie streaming services like Netflix, recommendation systems are important for helping users to discover new content to enjoy. While the details of this system are mostly confidential, what we do know is that it is a combination of various individual recommendation systems, including some systems which leverage collaborative filtering systems. In light of this, the problem we examine is movie recommendations through collaborative filtering.

Collaborative filtering is an approach for recommendation systems which relies on the ratings for particular user as well as the ratings of similar users. The underlying assumption is that if we can accurately predict movie ratings, then we can recommend new movies to users that they are likely to enjoy, including movies the user may not have considered before. Therefore, in the context of movie recommendation, collaborative filtering aims to predict unknown movie ratings for a particular user, based on that user's known ratings as well as the movie ratings by other users in the system. As opposed to content-based systems, collaborative filtering accounts for users with diverse taste, so long as there are other users with similar preferences. By finding similar users, new items can be recommended based on the assumption that items which are liked by similar users will be liked by the user in question.

There are many ways to perform collaborative filtering such as utilizing k-nearest neighbor clustering with user profiles [2]. Various approaches for measuring similarity have been proposed, but a simple approach is to represent a user profile as a vector, and then use some measure of similarity between those vectors (e.g., cosine similarity). An alternative k-nearest-neighbor approach instead computes similarity between pairs of items with the idea that users who like a particular item will like similar items [11]. Another common method for performing collaborative filtering is with matrix factorization [8]. With this technique a user-item matrix is factorized into two matrices with the inner dimension representing some latent factors. The resulting factorization represents both users and items in terms of the latent factors in such a way that new items can be recommended to users based on the latent factors.

Lately, deep learning has demonstrated its effectiveness in coping with recommendation tasks. Due to its state-of-the-art performances and high-quality recommendations, deep learning techniques have been gaining momentum in recommender system. Compared with traditional recommendation models, deep learning provides a better understanding of user's demands, item's characteristics and historical interac-

[1] techcrunch.com/2017/12/11/netflix-users-collectively-watched-1-billion-hours-of-content-per-week-in-2017/

[2] tvtechnology.com/news/netflix-surpasses-11-billion-in-2017-revenue

tions between them. We apply the deep learning approach for movie recommendation.

The rest of the paper is organized as follows. The most popular approaches for collaborative filtering are discussed in Section II. These methods work by computing neighborhoods of similar users or items. In contrast, in Section III we propose a deep learning approach for collaborative filtering based on an autoencoder. We demonstrate in Section IV that our approach outperforms the neighborhood-based baseline. We give a concluding remark in Section V.

## II. Related Work

The most common method of performing collaborative filtering is to utilize a k-nearest-neighbor approach between users [2]. With this technique, it first starts with a user-item matrix $R$, where $R_{i,j}$ gives the rating of user $i$ for item $j$ and the value 0 indicates that a particular rating is missing. From $R$ a user-user similarity matrix $S$ is computed, where $S_{i,j}$ is the similarity between user $i$ and user $j$, which can be computed with $R \cdot R^T$. Note that using other distance metrics, such as the correlation similarity measure or cosine similarity, to populate $S$ are also effective. Once $S$ is computed, we can predict the rating of user $i$ for item $j$ by computing $R_j^T \cdot S_i$, which essentially computes the average of the other users' ratings for item $j$ weighted by their similarity to user $i$.

We can also use the $k$ most similar users to user $i$ to predict the rating for item $j$. Empirically, this works better than the weighted average over all users, although some extra work is required at test time in order to compute the $k$ nearest neighbors. This approach relies on the assumption that if two users rated the same item similarly, they are likely to rate other items similarly as well. At scale, data structures such as ball trees and k-d trees (a binary space partition tree in k-dimensions) can be utilized to more efficiently compute local neighbors between user profiles.

An alternative k-nearest-neighbor approach instead computes similarity between pairs of items (as opposed to users) with the idea that users who like a particular item will like similar items [11]. With this approach we compute an item-to-item similarity matrix $I$ as $R^T \cdot R$. As before, we can also use other similarly metrics to populate $I$. In order to predict the rating for user $i$ on item $j$, we can compute $R_i \cdot I_j$, which gives an average of the ratings provided by user $i$ weighted by the similarity of those items to item $j$. Since there tends to be many more users than items in a recommender system, user-user collaborative filtering can be more performant.

Another common method for performing collaborative filtering is with matrix factorization [8]. With this technique a user-item matrix is factorized into two matrices with the inner dimension representing some latent factors using techniques such as singular value decomposition (SVD). The resulting factorization represents both users and items in terms of the latent factors in such a way that they can be used to recommend new items. As with item-item neighborhood approaches, our preliminary experiments on movie ratings indicate that user-user neighborhood approaches are superior to matrix factorization.

Deep learning has revolutionized many fields of computer science, including natural language processing [9]. Despite this, deep learning is relatively new in the area of recommender systems, and has not received much attention [18]. Having said that, Wang et al. [15] propose a collaborative deep learning (CDL) model which jointly performs deep representation learning for the content information and collaborative filtering for the ratings matrix. CDL is differed from ours, since the former relies on content information, whereas we do not. Elkahky et al. [3] introduce a deep learning recommendation system according to the web browsing history and search queries provided by users. They maximize the similarity between users and their preferred items by mapping users and items to a latent space. A constraint imposed on this approach is that browsing history and users search queries are required, which are not always available. Wei et al. [16] develop a deep neural network model which extracts the content features of items into prediction of ratings for cold start items. Our recommendation system is differed, since we do not deal with user content.

## III. Our Proposed Recommendation System

Deep learning, which is essentially just deep artificial neural networks, is able to learn complex decision boundaries for classification or complex non-linear regressions. By stacking large numbers of hidden layers in these networks, deep neural networks can learn complex functions by learning to extract many low level features from the data and composing them in useful non-linear combinations.

### A. Network Architecture

While neural networks are theoretically able to approximate any computable function, including the mapping from user profiles to movie ratings, in practice great care must be taken when selecting the architecture of the neural network. While the extracted structure of our network is subject to change, there are some reasonable starting places.

**Inputs**. The inputs to our network architecture are two $n$-dimensional vectors, where $n$ is the number of movies in a movie dataset, such as the MovieLens database. One vector encodes a particular user profile, with each dimension indicating the rating the user gave for a particular film (or a zero to indicate that no rating has been given). The other vector is a one-hot encoding of a particular movie (i.e., a vector with a single "hot" dimension set to 1, with all other values set to zero). These two vectors request that the network predict a rating for a particular user for a specific movie.

One advantage of this input format is that we can do without a single rating from a known user profile, and use the

known rating for withheld item as a labeled example. Consequently, even though we only have 270,000 user profiles, each one of the 26,000,000 individual ratings constitutes a train example.

**Hidden Layers**. There are a variety of ways to structure a simple feed-forward neural network. We start with a number of the standard fully-connected layers. However, we also experiment with alternative structures, such as ResNets [7], which currently obtain state-of-the-art results in other fields such as image recognition.

**Output**. There are two main possibilities for the output of our network. The first is to treat this problem as a classification problem, with five different class representing the five start ratings that are present in the data. Under this architecture, we treat the five outputs of our network as unnormalized log probabilities, and use cross entropy as our loss function.

With the basic neural network architecture introduced above, we describe the deep learning architecture proposed as an alternative to the user-based neighborhood approach. We first consider the dimensions of the input and output of the neural network. In order to maximize the amount of training data we can feed to the network, we consider a training example to be a user profile (i.e. a row from the user-item matrix $R$) with one rating withheld. The loss of the network on that training example must be computed with respect to the single withheld rating. The consequence of this is that each individual rating in the training set corresponds to a training example, rather than each user.

As we are interested in what is essentially a regression, we choose to use *root mean squared error (RMSE)* with respect to known ratings as our loss function. Compared to the mean absolute error, root mean squared error more heavily penalizes predictions which are further off. We reason that this is good in the context of recommender system because predicting a high rating for an item the user did not enjoy significantly impacts the quality of the recommendations. On the other hand, smaller errors in prediction likely result in recommendations that are still useful—perhaps the regression is not exactly correct, but at least the highest predicted rating are likely to be relevant to the user.

*1) Autoencoder:* One of the existing deep learning models is the Deep Neural Network (DNN) model. DNN is a Multi-Layer Perceptron (MLP) model with many hidden layers. The uniqueness of DNN is due to its larger number of hidden units and better parameter initialization techniques. A DNN model with large number of hidden units can have better modeling power. Although the learned parameters of the DNN model is a local optimal, which requires more training data and more computational power, it can perform much better than those with less hidden units. Deep Auto Encoder is a special type of DNN. (See Figure 1 for a sample autoencoder.)

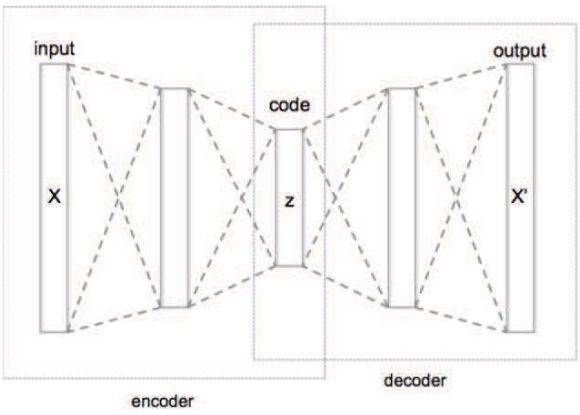


Figure 1. An autoencoder with three fully-connected hidden layers

An autoencoder is a neural network that is trained to copy its input to its output, with the typical purpose of dimension reduction, i.e., the process of reducing the number of random variables under consideration. It features an encoder function to create a hidden layer (or multiple hidden layers) which contains a code to describe the input. There is a decoder which creates a reconstruction of the input from the hidden layer. An autoencoder can then become useful by having a hidden layer smaller than the input layer, forcing it to create a compressed representation of the data in the hidden layer by learning correlations in the data. This autoencoder is a form of unsupervised learning, meaning that an autoencoder only needs unlabelled data, which is a set of input data rather than input-output pairs. Through an unsupervised learning algorithm, for linear reconstructions the autoencoder attempts to learn a function to minimize the root mean square difference.

To compute the *root mean square error* (RMSE) of a machine learning model, we can measure the performance of the model. RMSE is defined as

$$RMSE = \frac{1}{m}\sum_i (\hat{y} - y)_i^2, \quad \hat{y} = \mathbf{w}^T\mathbf{x} \tag{1}$$

where $\mathbf{w} \in \Re^n$ is a vector of parameters, $\mathbf{x} \in \Re^n$ is a vector used for predicting a scalar value $y \in \Re$, and $\hat{y}$ is the value that a machine learning model predicts what the scalar value $y \in \Re$ should be.

Note that RMSE decreases to 0 when $\hat{y} = y$ and the error increases when the *Euclidean distance* between the predicted values and the target values increases.

*2) Multilayer Perceptron:* Initially, the architecture of our recommender system consists of input from the row of the user-item matrix $R$ with the rating for some item $j$ withheld, along with a one-hot encoded query which indicates the network should predict the rating for user $i$ on item $j$. Unfortunately, this architecture has been proved difficult to train, since the network must learn to understand not only user profiles, but also the interplay between those profiles and the query inputs. With respect to the root mean squared

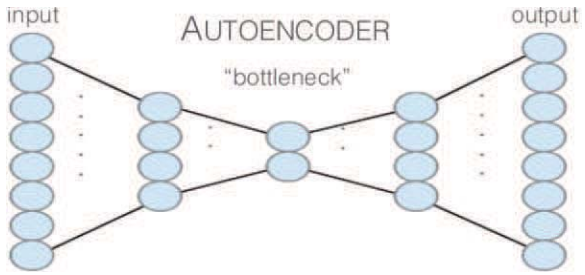


Figure 2. Overview of the network architecture of our recommender

error on the training data, we never achieved a loss less than 1.2 with this architecture.

Instead, we take inspiration from the concept of an *autoencoder* to design our neural network architecture (see Figure 2). This simple architecture takes an input and connects it to some number of fully connected hidden layers which include a "bottleneck." This bottleneck is a hidden layer which has a much smaller dimensionality than the input. The output of the network is then re-expanded to have the same dimensionality as the input. The network is then trained to learn the identity function, with the idea that in order for the network to compute the identity function through the bottleneck, it must learn a dense representation of the input. Thus, the autoencoder could be viewed as something akin to a *dimensionality reduction* technique. We can also hope that the bottleneck layer learns something useful related to the structure underlying the input. For example, a neuron in the bottleneck layer might represent something related to the genre of a movie or similar movie groupings.

Note that we are not interested in learning to compute an identity function—after all, our goal is to predict missing ratings, not reproduce the zeros in the input vectors. Consequently, while our final network architecture resembles an autoencoder with the bottleneck hidden layers and the matching dimensions on input and output, the network is actually trained using a loss function for regression (i.e., RMSE) with the aim of learning to predict missing ratings.

More specifically, the training examples to the network are user profiles with one rating withheld, and the output is the predicted ratings for all movies in the dataset. While the network is expected to predict ratings for each movie based on a user profile, we only have the answer for the one withheld rating. Consequently, we only propagate loss for the missing rating when learning from the training example.[3]

*3) The Deep Learning Recommender System:* Withholding ratings does have the unfortunate consequence that our deep learning model is only able to learn ratings for movies similar to what the user has actually watched, as the loss function is not directly affected by the output on unrelated movies. Due to the bottleneck layer, the model is required to generalize to some degree, but the model may have difficulty for movies which are drastically different than the movies the user actually rated. While users do watch movies they rate lowly, most of the time they do not rate more than a few hundred items, and avoid watching completely non-relevant movies, so it may be difficult for the model to predict ratings for completely unrelated movies.

For the purposes of our loss function, which is *root mean squared error* on known ratings, the fact that our network may not learn how to output ratings for completely unrelated movies does not seem to affect the test loss, probably because the movies in the test data are related enough that the patterns learned from the training data generalize to the ratings in the test data. Of course, it may affect the rankings, so it could be desirable to add a *regularization* term (discussed in details in Section III-B) to the loss.

With this basic design in place, we have experimented with several variations of this architecture using various numbers of layers, and various sizes for the bottleneck layer. The most interesting parameter was the size of the smallest bottleneck layer, and after experimenting with various values, we eventually settled on a bottleneck size of 512. From there we experimented with different numbers of fully connected layers, always using powers of 2 to increase and decrease the dimensionality. The final network topology has seven fully connected hidden layers with dimensions [4096, 2048, 1024, 512, 1024, 2048, 4096]. Each layer used a rectified linear unit[4] as the non-linear activation function. The connecting weights of the hidden layers were initialized using Xavier initialization [4] with the biases set to zeros.

*4) Clustering:* We have considered the idea of using the smallest bottleneck layer in the network as some form of a natural clustering. By forcing the input into such a small dimensional space, the model must necessarily learn something about the underlying structure of the input data. The hypothesis was that by fixing a single neuron in the bottleneck layer and zeroing out the remaining neurons in the bottleneck layer, and then optimizing the input space for this particular activation, we can visualize that structure by showing the movies which trigger each *cluster*. For example, we expect that there might be a neuron or small set of neurons which trigger for various genres of movie, or various styles of filmography.

Table I gives an example of such a "cluster" from optimizing the input to trigger a single bottleneck neuron. These movies have common theme. Obviously, for this network to be able to accurately predict movie ratings it must learn some sort of structure. However, this structure is more distributed throughout the bottleneck layer than expected. One potential solution to this problem is to add a regularization term to the loss which encourages sparsity in the bottleneck layer.

[3]In code, this can be accomplished with the tf:gather function.

[4]The rectified linear unit, or ReLU, is defined as $max(0; x)$. While simple, it is currently the state-of-the-art in activation functions for DNN.

Table I
A CLUSTER WHEN OPTIMIZING THE INPUT TO TRIGGER A SINGLE BOTTLENECK NEURON

| |
|---|
| Jules and Jim (Jules et Jim) (1961) |
| Frankenstein Must Be Destroyed (1969) |
| Lolita (1962) |
| Lawnmower Man, The (1992) |
| First Knight (1995) |
| Urban Legends: Final Cut (2000) |
| Fair Game (1995) |
| Guinevere (1999) |
| Paradine Case, The (1947) |
| 400 Blows The (Les quatre cents coups) (1959) |

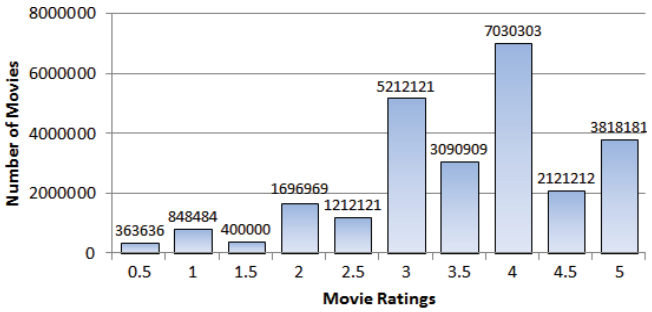


Figure 3. Distribution of ratings in the full MovieLens dataset

### B. Regularization

Regularization in deep learning, and in machine learning in general, is an important concept which solves the overfitting problem. It is very important to implement the regularization while training a good model, since it is a technique used in an attempt to solve the overfitting problem.

As mentioned earlier, regularization is an attempt to correct for model overfitting by introducing additional information to the cost function. Within the context of least squares linear regression, the regularization term is added to a standard least squares linear regression cost function $J$ as defined below.

$$J(\Theta) = \frac{1}{2}m\ [\sum_{i=1}^{m}(h_\Theta(x^i) - y^i)^2 + \lambda\sum_{j=1}^{n}\Theta_j^2\ ] \qquad (2)$$

where $\Theta$ is the parameter values, $m$ is the number of training examples with $n$ different features, $h_\Theta(x^i)$ is the estimator $h_\Theta$ value for the training example $i$, $y^i$ is the actual labeled value of training example $i$, and $\lambda$ is the *regularization constant*.

In discussing regularization we have employed L2 regularization, whereas L1 regularization is another such strategy for controlling overfitting. The two regularizations share the same goal but differ in a few key respects. Note that in Equation 2,

$$\lambda\sum_{j=1}^{n}\Theta_j^2 \qquad (3)$$

is the L2 regularization term, whereas in L1, the same regularization term is written as

$$\lambda\sum_{j=1}^{n}|\Theta_j| \qquad (4)$$

Hence, the difference between L1 and L2 is that L2 uses the sum of the square of the parameters, whereas L1 is the sum of the absolute value of the parameters. In essence, L1 regularization reduces some parameters associated with a given feature to zero, whereas L2 regularization does not set feature parameters to zero, but will only continue to reduce the value of a given $\Theta$.

## IV. EXPERIMENTAL RESULTS

Prior to presenting the experimental results of our recommendation system, we discuss the dataset used for the empirical study and the experimental setup. We first describe the MovieLens dataset and then briefly explain the baseline model used as a point of comparison.

### A. MovieLens Data

In academia the most well-known movie ratings dataset is undoubtedly the MovieLens dataset [6], although a close second is probably the Netflix prize data released via Kaggle.[5] For our recommendation system we utilize the latest version of the MovieLens dataset, which is the recommended version for education and development.[6]

The MovieLens dataset is provided by GroupLens, which is a social computing research lab at the University of Minnesota. The full MovieLens dataset contains ratings for 45,115 movies provided by 270,896 different users. In total, the dataset contains 26,024,289 individual movie ratings. Each rating allows users to assign between half a star and five stars to a movie, in half star increments. Figure 3 shows the distribution of the ratings in the data. Each rating is also accompanied by a time stamp. Since the dataset does not contain a standard train/test split, we used these time stamps to split the data into training and test sets, with the oldest 90% of the data making the training set and the newest 10% of the data composing the test set. We did this with the intent to mimic the problem faced by real world movie recommendation systems which have all of the data up to a certain point in time, and are faced with predicting movie ratings going forward in time.

### B. Full Dataset Versus BaseLine

As previously mentioned, there are a number of popular methods for performing collaborative filtering, including nearest-neighbor based technique comparing user-user similarity [2], nearest-neighborhood comparing item-item similarity [11], and matrix factorization techniques [8]. We

[5]https://www.kaggle.com/netfix-inc/netfix-prize-data
[6]https://grouplens.org/datasets/movielens/latest

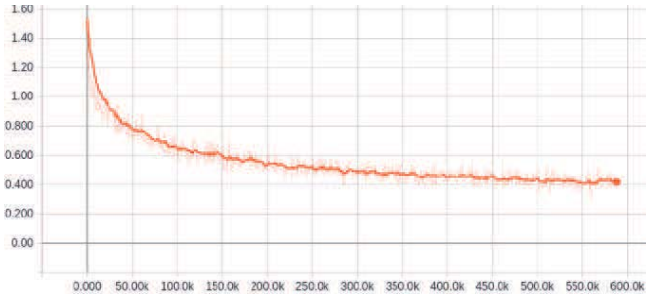


Figure 4. Graph showing loss (root mean squared error) decreasing over time. Each step represents 1,000 training examples

determined user-user neighborhood approach with cosine similarity and a neighborhood size of five performs the best with respect to root mean squared prediction error. In our empirical study, we used them all on the full MovieLens dataset. We allocated enough RAM to fully vectorize these algorithms. For example, in order to process the vectorized version of the user-user nearest neighbor approach, we computed a user-user similarity matrix which took nearly 600 GB in RAM. The non-vectorized brute force version of the algorithm required more than a week to finish. An alternative is to utilize a small version of the MovieLens dataset, called the BaseLine dataset, which contains only 943 users and 1,682 movies as a development dataset. The BaseLine database can be split into a train/test set, and we can measure the root mean squared error of the predictions of each of the proposed baseline algorithms.

### C. Results

Using 90% of the full MovieLens dataset as training, we trained the architecture described in Section III-A. It took roughly 4 days using a Titan X GPU to make 30 passes over the entire data before the training loss stabilized. Figure 4 shows the training loss (i.e. RMSE) decreasing over time.

We discuss the results of our model on the test set and compare its results to the user-based neighborhood models.

*1) Root Mean Squared Error:* Table II summarizes the results comparing our model-based approach with the user-based neighborhood baseline. On the training data, our approach is stabilized around 0.42. The neighborhood approach has learned parameters, as it simply relies on the training data itself to make predictions. Consequently, there is no training loss to report.

Table II
ROOT MEAN SQUARED ERROR (RMSE) FOR OUR USER-BASED NEIGHBORHOOD BASELINE AND AUTOENCODER INSPIRED BY OUR MODEL-BASED APPROACH

| | User-User KNN | Model-based |
|---|---|---|
| Train | N/A | 0.4209 |
| Test | 11.6715 | 0.3544 |

On test data, our deep learning model-based algorithm outperforms the neighborhood approach by a large margin. However, it should be noted that for the purpose of making movie recommendations, we do not actually care about the error. Instead what we care about is the ranking of the top few most highly rated movies. It is not an unreasonable assumption that the algorithm which ranks *better* will also have *lower* root mean squared error, but it is entirely possible that despite the higher errors, the top ranked movies from the model-based approach produce superior recommendations. This is especially true when we consider that our algorithm does not directly learn about highly unrelated movies.

*2) Comparing Our Movie Recommendation Systems with Others:* Besides using RMSE as shown in Table II, we compare between various well-known movie recommenders and our deep learning movie recommendation model. These existing movie recommenders were chosen, since they achieve high accuracy in recommendations on movies based on their respective model, and more importantly they are simply based on user ratings, but not solely on contents.

- **MF**. Yu et al. [17] and Singh et al. [13] predict ratings on movies based on matrix factorization (MF), which can be adopted for solving large-scale collaborative filtering problems. Yu et al. develop a non-parametric matrix factorization (NPMF) method, which exploits data sparsity effectively and achieves predicted rankings on items comparable to or even superior than the performance of the state-of-the-art low-rank matrix factorization methods. Singh et al. introduce a collective matrix factorization (CMF) approach based on relational learning, which predicts user ratings on items based on the items' genres and role players, which are treated as unknown values of a relation between entities of a certain item using a given database of entities and observed relations among entities. Singh et al. propose different stochastic optimization methods to handle and work efficiently on large and sparse data sets with relational schemes. They have demonstrated that their model is practical to process relational domains with hundreds of thousands of entities.
- **ML**. Besides the matrix factorization methods, probabilistic frameworks have been introduced for rating predictions. Shi et al. [12] propose a joint matrix factorization model for making context-aware item recommendations. The matrix factorization model developed by Shi et al. relies not only on factorizing the user-item rating matrix but also considers contextual information of items. The model is capable of learning from user-item matrix, as in conventional collaborative filtering model, and simultaneously uses contextual information during the recommendation process. However, a significant difference between Shi et al.'s MF model and other MF approaches is that the contextual information of the former is based on movie mood, whereas other MF models makes recommendations according to the

contextual information on movies.

- MudRecS [10], which makes recommendations on books, movies, music, and paintings similar in content to other books, movies, music, and/or paintings, respectively that a MudRecS user is interested in. MudRecS does not rely on users' access patterns/histories, connection information extracted from social networking sites, collaborated filtering methods, or user personal attributes (such as gender/age) to perform the recommendation task. It simply considers the users' ratings, genres, role players (authors or artists), and reviews of different multimedia items. MudRecS predicts the *ratings* of multimedia items that match the interests of a user to make recommendations.
- **Netflix**. We compare our deep learning recommendation system indirectly against the 20 systems that participated in the Netflix contest in 2008 through MudRecS [10]. The open competition was held by Netflix, an online DVD-rental service, and the Netflix Prize was awarded to the best recommendation algorithm with the lowest RMSE score in predicting user ratings on films based on previous ratings. On September 21, 2009, the grand prize of one million dollars were given. The RMSE scores achieved by each of the twenty systems, as well as detailed discussions on their rating prediction algorithms, can be found on the Netflix website.[7]

Figure 5 shows the Mean Absolute Error (MAE) and RMSE scores of our deep learning movie recommender and other recommendation systems on the MovieLens dataset. RMSE and MAE are two performance metrics widely-used for evaluating rating predictions on multimedia data [1]. Both RMSE and MAE measure the *average magnitude* of *error*, i.e., the average prediction error, on incorrectly assigned ratings. The error values computed by RMSE are squared before they are summed and averaged, which yield a relatively *high* weight to errors of *large* magnitude, whereas MAE is a *linear* score, i.e., the absolute values of individual differences in incorrect assignments are weighted equally in the average. Our deep learning recommender outperforms each of the movie recommenders as shown in Figure 5, and the RMSE and MAE values are statistically significant ($p < 0.01$) [14].

On the *Netflix* dataset, MudRecS achieves a RMSE score[8] of 0.8571. MudRecS outperforms 18 recommendation systems and is only outperformed by two systems (Belkor and Ensemble), both of which achieve the same score of 0.8567, a small, insignificant fraction (0.8571 - 0.8567 = 0.0004) better than MudRecS. The reason for the slightly better RMSE score achieved by the two systems on the Netflix dataset are twofold. Unlike MudRecS, Belkor and Ensemble

[7]https://www.netflixprize.com/leaderboard.html

[8]MAE scores were not computed on the Netflix dataset due to their unavailability for the other 20 recommenders.

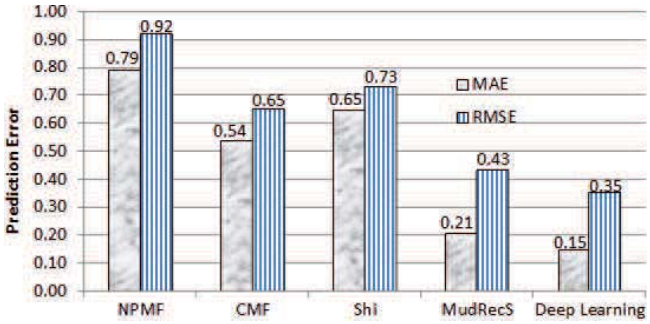


Figure 5. The MAE and RMSE scores for various movie recommendation systems based on the MovieLens dataset

were specifically designed for movie rating predictions, and the construction of their algorithms focus on rating patterns found in movies which may not apply to other domains. Moreover, Belkor and Ensemble account for temporal effects, i.e., the fact that a user's preference changes over time, which may lead to different ratings for the same movie over time. The temporal effect, however, does not apply to all users and requires a larger subset of training data in order to obtain reliable results, which are the constraints. In considering a 95% confidence interval, MudRecS significantly outperforms 17 recommendation systems and is *not* significantly outperformed by *any* of the twenty systems. CineMatch, Netflix's recommender, achieves an RMSE score of 0.9514 on the Netflix dataset, which is outperformed by MudRecS. We ran our deep learning recommender system on the Netflix dataset and achieves a 0.782 RMSE score, which is lower than MudRecS, even though the results are not statistically significant. However, our recommender performs at least as good as MudRecS based on the Netflix dataset.

*3) User Evaluation:* In order to establish the usefulness of our deep learning approach in making movie recommendation, we conducted a user study in which users, who play the role of appraisers, had the chance to evaluate movies recommended by our system and the user-based neighborhood (KNN) approach.

Appraisers were shown a user profile, which consisted of every movie the corresponding user had rated, as well as the associated ratings. Each appraiser was then presented two possible recommendations: one from our system and one from the user-based neighborhood approach. The recommendations were chosen by picking the movie with the highest predicted rating from either system, excluding movies that had already been rated by the user. The order in which the two possible recommendations were shown was randomized. Appraisers were asked to pick which recommendation they thought was more relevant to the given user profile (see Figure 6 for an example of the study).

A total of 100 participants, who were students at the authors' university, were used in the study. Each user, who


Figure 6. An example of the type of questions appraisers were asked to answer in the user evaluation of our deep learning-based system and the user-based KNN approach.

is an appraiser, was asked to rate 15 randomly chosen recommendations. In this survey, 71.67% of the time appraisers preferred the recommendation made using our deep learning approach over the recommendation made by the baseline approach, and this result is encouraging. Of course, it is clear that this survey using a small sample size. In addition, most of the appraisers indicated that they were unfamiliar with most of movies referenced in the survey. Realizing this problem in advance, we indicated in the survey that they were allowed to use resources like Google[9] and IMDBa[10] while making their judgements.

## V. Conclusions

We have proposed a simple neural network model which performs well in terms of root mean squared error for collaborative filtering. This adds to existing literature which suggests that deep learning can be a powerful tool for a variety of problems in information retrieval [18]. In the end, this work makes improvement in terms of predicting ratings of and recommending movies for users. Our recommender system applies regularization to further minimize the prediction errors. In addition, our system was able to handily outperform the neighborhood-based baseline, and was able to provide superior movie recommendations. As an added advantage of our deep learning approach, it is much more scalable at test time.

## References

[1] T. Chai and R. Draxler. Root Mean Square Error (RMSE) or Mean Absolute Error (MAE)? *Geoscientific Model Development Discussions*, 7(1):1525–1534, 2014.

[2] W. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Addison Wesley, 2010.

[3] A. Elkahky, Y. Song, and X. He. A Multi-View Deep Learning Approach for Cross Domain User Modeling in Recommendation Systems. In *WWW*, pages 278–288, 2015.

[4] X. Glorot and Y. Bengio. Understanding the Difficulty of Training Deep Feed-Forward Neural Networks. In *AISTATS*, pages 249–256, 2010.

[9]https://www.google.com
[10]www.imdb.com

[5] C. Gomez-Uribe and N. Hunt. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM TMIS*, 6(4):Article 13, 2016.

[6] F. Harper and J. Konstan. The Movielens Datasets: History and Context. *ACM TiiS*, 5(4):Article 19, 2016.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *IEEE CVPR*, pages 770–778, 2016.

[8] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, 2009.

[9] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. Alsaadi. A Survey of Deep Neural Network Architectures and Their Applications. *Neurocomputing*, 234:11–26, 2017.

[10] R. Qumsiyeh and Y.-K. Ng. Predicting the Ratings of Multimedia Items for Making Personalized Recommendations. In *ACM SIGIR*, pages 475–484, 2012.

[11] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based Collaborative Filtering Recommendation Algorithms. In *WWW*, pages 285–295, 2001.

[12] Y. Shi, M. Larson, and A. Hanjalic. Mining Mood-Specific Movie Similarity with Matrix Factorization for Context-Aware Recommendation. In *Context-Aware Movie Recommendation*, pages 34–40, 2010.

[13] A. Singh and G. Gordon. Relational Learning via Collective Matrix Factorization. In *SIGKDD*, pages 650–658, 2008.

[14] M. Smucker, J. Allan, and B. Carterette. Agreement Among Statistical Significance Tests for Information Retrieval Evaluation at Varying Sample Sizes. In *SIGIR*, pages 630–631, 2009.

[15] H. Wang, N. Wang, and D.-Y. Yeung. Collaborative Deep Learning for Recommender Systems. In *KDD*, pages 1235–1244, 2015.

[16] J. Wei, J. He, K. Chen, Y. Zhou, and Z. Tang. Collaborative Filtering and Deep Learning Based Recommendation System for Cold Start Items. *Expert Systems with Applications*, 69:29–39, 2017.

[17] K. Yu, S. Zhu, J. Lafferty, and Y. Gong. Fast Nonparametric Matrix Factorization for Large-Scale Collaborative Filtering. In *ACM SIGIR*, pages 211–218, 2009.

[18] S. Zhang, L. Yao, and A. Sun. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM JOCCH*, 1(1):Article 35, 2017.