APPLYING DEEP NEURAL NETWORKS TO MOVIE RECOMMENDATION

A Project Submitted to the Faculty of
UNIVERSITY OF MINNESOTA
by

Utkarsh Kajaria

In Partial Fulfillment of the Requirements for the Degree of
Master's of Science
in
Data Science

Chaired by

Joseph Konstan (advisor)
Singdhansu Chatterjee
Haiyi Zhu

June 2017

Acknowledgement

I would like to express my sincerest gratitude to Prof. Joseph Konstan for guiding me throughout this project and for being patient and encouraging as I explored an uncharted territory. I would like to thank Prof. Singdhansu Chatterjee and Prof. Haiyi Zhu for taking the time to review my work and being a part of my final exam committee. I would also like to thank the Grouplens Research Lab for making the data available for me to conduct this work and Max Harper for helping me make use of it in the best way possible.

Abstract

Recommender systems are the workhorse that shape much of our experiences online: from suggesting friends on social networks to product recommendations on e-retail websites. With ever increasing choices to select from and demand for greater peronalization of content, newer ways of solving the problem of recommendation are warranted. The success in training deep neural networks for image classification recently, has established it as a promising framework that works by creating a feature heirarchy to discover latent structures from large, high-dimensional datasets -a task that is reminiscent of the latent-factor based recommender systems. This begs the question: Can deep learning be used as a new approach to solve the recommendation problem? In this project we try to answer this question by formulating the task of movie rating prediction as a multi-class classification problem and conducting experiments to evaluate the quality of predictions given by deep neural networks when provided with the history of users' ratings along with the metadata of asssociated items.

Contents

1 Introduction

The advent of internet and particularly mobile computing and e-commerce in the last decade brought about a data deluge. With the problem of finding what we want from enormous options in a limited amount of time, the requirement of having a recommender system cannot be overstated. Indeed, the autonomous nature of recommendations based on user interactions with a website has led the trasition of user experience to be highly dynamic and more meaningful.

It is not surprising that we have witnessed substantial attention being given to advance the state of the art of recommender systems along the same timeline. The Netflix challenge stirred up a lot of interest on this topic and encouraged some of the best machine learning and data mining researchers to work on solving optimization problems that resulted in more accurate movie predictions for users.

Recently, researchers had success in training deep neural networks (i.e. with more than a single hidden layer)[1] for image classification [14]. Not only has this caused a lot of excitement in the machine learning community, it is also an opportunity for areas of research that have used the ML framework, to revisit the problems with a new tool and a fresh perspective. In this report, we present our approach to applying deep learning to the problem of movie recommendation. Specifically, we aim to explore if deep learning methods can be used to learn from users' rating history and additional metadata available from Movielens system to provide movie rating predictions for users. We evaluate the quality of predicted user ratings using Mean Absolute Error (MAE) and compare our results against the benchmark of SVD, a popular collaborative filtering model.

2 Background

Before the 2000s there were two mainstream types of methods of recommender systems -Collaborative Filtering and Content Based. In this section, we briefly discuss the basic idea behind these two methods. Subsequently, we provide an intuition behind applying neural network to the problem of movie recommendation.

2.1 Collaborative Filtering

Collaborative filtering systems aim to suggest new items or predict the utility of a certain item for a particular user based on users past liking and the opinions of other like-minded users [12]. It works on the premise that users with similar preference for items tend to agree again in the future on different items. These can be categorized into two types: Neighborhood or latent factors based.

[1] `https://deeplearning4j.org/neuralnet-overview#concept`

2.1.1 Neighborhood Based

Collaborative filtering systems of this type can be further divided into two types based on whether the the notion of neighborhood is between a pair of users or a pair of items.

2.1.1.1 User Based

The idea is to the predict the rating of an item for a user based on the opinion of similar users. Here similar can refer to a statistical metric such as Pearson correlation and the opinion can be represented as a function (e.g. average) of the ratings given to that item by the similar users.

2.1.1.2 Item based

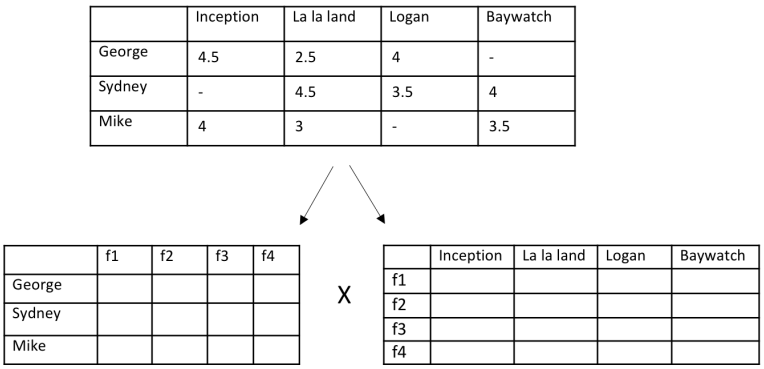
The user based collaborative filtering performs poorly when magnitude of items is very large compared to the ratings. In addition, computing similarities between all pairs of users is computationally expensive and not suitable for real world applications. The item-item based collaborative filtering first proposed by Sarwar et al in 2001 [31] was a major breakthrough as it overcame this problem. Instead of computing similarity between users, the algorithm finds items that are similar to other items (which can be computed offline) that a user is known to like, aggregates those items and then recommends the most popular or correlated items. The similar items here, are other items that tend to get similar ratings when rated by the same user. The item-item algorithm was first deployed on a large scale by Amazon [24].

2.1.2 Latent Factor Based

As the name suggests, this method provides recommendations by learing certain inherent characteristics of items, and user's preference for these characteristics. These characterstics may be relatable factors such as quirkiness or depth of a movie character, which otherwise is not present as an explicit feature or it could be something completely uninterpretable [20].

Matrix Factorization is one of the most popular latent based methods. As the name suggests it refers to decomposing a matrix into two factor matrices which when multiplied would give us back the original matrix. To put this into perspective, lets say we have a group of users, a set of items (movies) and the ratings that the users have given to some of the items. We would like to predict how the users would rate the items that they have not yet rated, such that we can make recommendations to the users.

The task of prediction then requires that the blanks in the ratings table below be filled with values which would be consistent with the values originally present.



The intuition behind splitting this sparse matrix is that there should be some latent features that determine how a user rates an item. Hence, if we discover these latent features, $f1-f4$, we can represent users preference for those features in one matrix and the presence of those features in items with another matrix. Then we should be able to multiply these two matrices to arrive at the denser version of the original matrix i.e. essentially predict the missing values in the ratings table.

2.2 Content based

Collaborative filtering techniques have been very popular, however, they suffer from the cold start problem i.e. when a user is new and has not rated any item, these models cannot provide meaningful recommendations. Content-based recommenders are more effective in such a scenario.

The reason behind their robustness is due to the fact that content-based systems derive the similarity between items based on their content rather than how other users have rated the item. Lops et al. describe the steps involved [25] as follows:

a) Feature Extraction i.e. using information retrieval techniques (e.g. *tf-idf, n-grams*, etc) to generate feature rich vectors describing the item (e.g. movie title, genre etc).

b) User Profile Generation i.e. Construction of user profiles indicating their preferences. A user profile $\boldsymbol{U_i}$ for user u_i can be contructed by using machine learning techniques on a set of pairs of the form $< \boldsymbol{V_j}, r_{ij} >$ where r_{ij} is feedback given by user u_i to item v_j for j different items. This feedback can be either explicit i.e. when the user is asked to give their feedback (e.g. thumbs up or a star rating) on their entry into the system, or implicit, where the feedback is derived from analyzing the user's activity (e.g. clicks or views).

c) Filtering: Given a new item v_k, we can now predict how interested would the user u_i would be in it by comparing it's features $\boldsymbol{V_k}$ to this user's profile $\boldsymbol{U_i}$.

Based on this, we can come up with a strategy to provide top-k recommendations to the user.

Even though content-based is better than CF in context of cold-start, using them in isolation makes it susceptible to becoming a specialized recommeder with little novelty. In addition, these models are limited to the set of explicit features or those extracted through information filtering as opposed to CF which detects features latent within the items. Indeed, researchers have adopted *hybrid* approaches that combine different techniques to provide recommendations.

2.3 Hybrid Recommenders

Hybrid recommenders are a class of recommendation systems that combines different recommendation methods with the aim of overcoming limitations of using just one of them. As discussed by Burke [6], this combination can be done in different ways: separation of implementation of algorithm and combining the results, utilizing content based approach in collaborative filtering, utilizing collaborative filtering in content-based approach, constructing a model that systematically unified both approaches. In this section, we dicuss the various types of hybrid recommenders.

2.3.1 Weighted hybridization

These types of systems take a linear combination of the results of multiple recommenders to arrive at a prediction score for an item or a top ranked list of recommendations. The P-Tango system by Claypool et al. [7] is an example of such a system that starts of by giving equal weight to collaborative and content-based components and tunes them based on the accuracy of the predictions. The advantage of this technique lies in its simplicity. This, however, comes at the cost of assumption that the relative value of different techniques is uniform across all the possible items. This is not always true e.g. a collaborative recommender will be weaker for items that have few raters.

2.3.2 Switching Hybridization

As the name suggests, this method switches between different recommenders based on a heuristic about the ability of its constituent recommenders to provide good predictions in a given problem. Dailylearner by Billsus & Pazzani [4] is such a hybrid recommender composed of both collaborative and content based parts where it uses the former when the latter cannot make a recommendation with sufficient confidence. In doing so, it is able to provide out-of-the-box recommendation ability instead of specializing to the items similar to those previously rated highly by the user. However, its disadvantage is that the switching ability comes at the cost of additional parameterization for determining the switching criteria.

2.3.3 Mixed

Mixed hybridization can be applied when we need to make a lot of recommendations simultaneously. These systems combine recommendation results of different recommendation techniques at the same time instead of having just one recommendation per item. The PTV system by Smyth and Cotter [37] combines both content based and collaborative approaches to come up with a TV viewing schedule i.e. the final prediction is the aggregate of multiple recommenders suggesting individual programmes.

2.3.4 Feature combination

In this type of hybridization, the result from collaborative filtering is added as a feature to content-based recommenders to give the final recommendation. The advantage of this technique is that it does not rely exclusively on the collaborative data. Pipper by Basu et al [1] is an example of this type of this combination technique.

2.3.5 Cascade

In cascading hybridization, the recommendations of one technique are refined by another recommendation technique and this process is done iteratively to construct an order of preference for items for a particular user. This allows us to apply the higher priority recommender first and then use the lower priority recommender to fine tune this result. EntreeC by Burke [6] is an example of an hybridization that cascades knowledge-based and collaborative recommenders.

2.3.6 Feature Augumentation

In this approach, one technique is used to produce a rating or classification of an item and that information is then incorporated into the processing of the next recommendation technique. This is different from feature combination where raw data from different sources is combined. It is also different from cascading as in the latter we do not use the output of the first recommender, we simply refine the order of preference further using a new criterion. UseNet news filtering system by Sarwar et al. [32] uses filterbots which take into account the number of spelling errors and the size of included messages and add ratings to the database of ratings used by the collaborative part of the system, as artificial users to improve email filtering. The advantage of Augmentation is that it offers a way to improve the performance of a core system, e.g. GroupLens Recommendation Engine without modifying it.

2.3.7 Meta-level

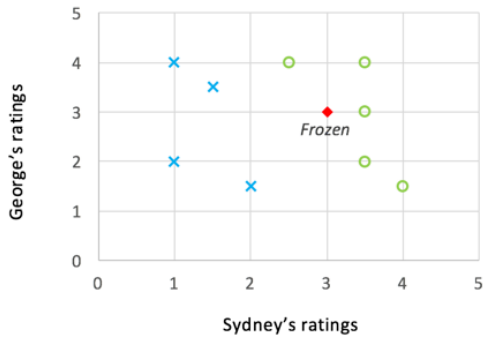
In contrast to feature augmentation, where we use a recommender model to construct feature for another recommender model, meta-level hybrid technique uses the entire first model as the input to the second model. The advantage of

this technique is, that the learned model is a richer, compressed representation of users interests, and hence a better starting point for collaborative filtering than the sparser, raw dataset. An example of meta-level technique is LaboUr by Schwab et al. [33] which uses instance-based learning to create content-based user profile that is then compared in a collaborative manner.

2.4 Neural Networks

Neural networks are systems modeled after human brain which are used to learn complex models in machine learning. We will develop an intuition for the application of neural networks to the problem of movie recommendation with the help of an example similar to [22].

Suppose we have a user who is contemplating watching the movie Frozen. Her friend George and Sydney saw this movie recently and upon asking their feedback on it, they both said they would give it a rating of 3 out of 5 stars which is not very convincing. The problem statement, then, is to determine whether a rating of 3 from them means a yes or no for the user by analyzing how much their suggestions have aligned with the user's liking in the past.



The above figure represents the history of ratings given by George and Sydney where the blue crosses represent the movies that user did not like so much and the green circles being the one he really liked. We can then use this information to come up with an expression for the users preference as a function of George and Sydneys ratings.

Formally, we use linear weighted combination of George and Sydneys ratings and compute a decision function like the following:

$$h(x_1, x_2, \theta_1, \theta_2, b) = \theta_1 x_1 + \theta_2 x_2 + b$$

where θ_1 and θ_2 are respectively the weights, or rather, the importance that we assign to the ratings of George (x_1) and Sydney (x_2) respectively to come up with a decision for our user. Using matrix notations we can write:

$$h(\mathbf{x}, \theta, b) = \theta^T \mathbf{x} + b$$

Further, to keep the above function from taking arbitrarily large or small values and force it to be in the range [0,1] we feed it through the sigmoid function such that:

$$h(\mathbf{x}, \theta, b) = g(\theta^T x + b) \text{ where } g(y) = \frac{1}{1 + e^{-y}}$$

This will help us approximate the output of the decision function as true or false i.e. the user should watch the movie or not watch it respectively.

To completely define the decision function, we use stochastic gradient descent to remove the arbitrariness from the variables (θ and b) that we have written in the above equation.

Stochastic gradient descent is an optimization procedure which makes small changes to these variables in multiple steps, or *iterations* with the aim of minimizing the *cost function*, which is a function that is representative of the extent of errors a decision function makes. Since we want to have as few errors as possible i.e. minimize the value of the cost function, the algorithm updates these variables in the opposite direction to the gradient of the objective function. The size of each step is controlled by a parameter called *learning rate*.

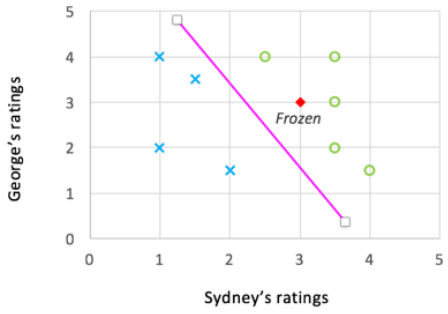
Formally, we want to minimize the following objective function:

$$J(\theta, b) = (h(x^{(1)}, \theta, b) - y^{(1)})^2 + (h(x^{(2)}, \theta, b) - y^{(2)})^2 + ... + (h(x^{(n)}, \theta, b) - y^{(n)})^2$$

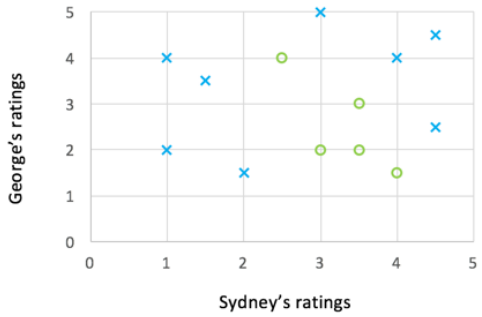
$$= \sum_{i=1}^{n} h(x^{(i)}, \theta, b) y^{(i)})^2$$

where $(h(x^{(i)}, \theta, b) y^{(i)})^2$ represents the square of the difference between the output of the decision function h and $y^{(i)}$, the original binary label representing our liking for the ith movie.

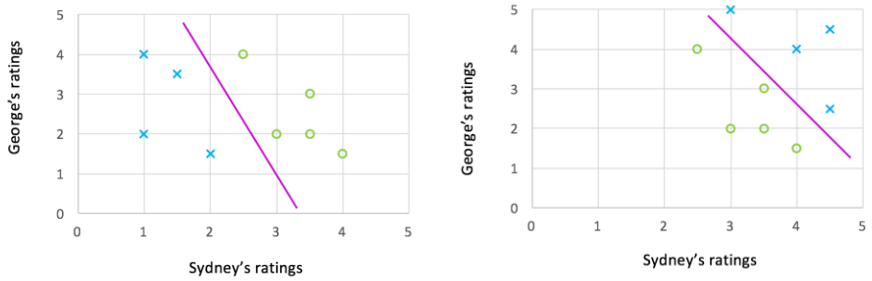
Finally, we use our decision function to determine whether the user watch a movie or not. Clearly, chances are that she probably is going to like watching *Frozen*.



However, the above approach worked on this carefully constructed example as the data was linearly separable. Next, we examine the case where the users taste was more nuanced as shown below:



In this case, a linear decision boundary of the manner we used before would not do very well as a decision function. We can, however, split this data into two subsets that are linearly separable and compute two decision functions $h_1(x, \theta_1, \theta_2, b_1)$ and $h_2(x, \theta_3, \theta_4, b_2)$ as shown below:



We can now compute a decision function by using a linear weighted combination of these decision function in a manner very similar to our initial method.

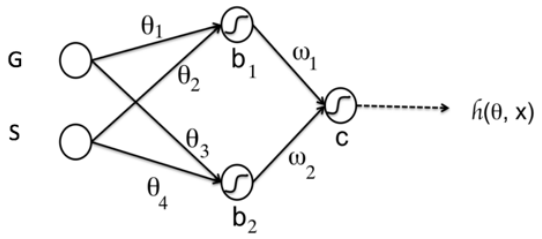
$$h(x) = g\Big(w_1 h_1(x, \theta_1, \theta_2, b_1) + w_2 h_2(x, \theta_3, \theta_4, b_2) + c\Big)$$

In the general case, instead of partitioning a problem, we use the stochastic gradient descent and *backpropagation algorithm* to compute $w_1, w_2, \theta_1, \theta_2, \theta_3, \theta_4, b_1, b_2$ and c simultaneously.

The backpropagation is the work-horse of a neural network. It is an algorithm that makes the simultaneous computation of the above weights. The training of a network happens as follows: the input is propagated through the input layer to hidden layers and finally the output layer. This is called the forward pass. Due to the initial weights applied to the network, we get an output. This output value is compared against the expected output to compute the error.

Next, we use this error to calculate gradients for each weight in the layer closest to the output layer which we use to update these weights so as to minimize the error. Next we use these updated wieghts to recalculate weights for the layer above and do so for each layer till we reach the input layer. This is called the backward pass. This propagation of changes is made possible due to the *chain rule* of differentiation. We continue doing forward and backward passes till the point we reach a low enough error in the output.

The above problem of prediction of user's movie score using that of her two friends' essentially represents a neural network with one hidden layer containing two neurons. The first neuron computes values for function h1 and the second neuron computes values for function h_2.



The sigmoid function that maps real value to bounded values between 0, 1 is called the *activation function*. Activation functions take in weighted data and output non-linear transformation of the data, a property which makes neural networks powerful. In order to use backward propagation, the activation function must be differentiable and possibly bounded. The parameters inside the network θ, ω are called weights whereas b_1, b_2 and c are called biases.

The field of artificial neural networks can be traced back to 1950s when the *Perceptron* algorithm was developed by Rosenblatt [29]. In the late 1970s, researchers discovered that a Perceptron fails to learn many nonlinear decision functions, XOR being an embarrassing example. Later, in the 1980s, it was found that stacking multiple layers of linear classifiers (essentially a multi-layer perceptron) approximated nonlinear decision functions. However due to the vanishing gradient problem, neural networks did not see widespread success.

Interest in deep feedforward networks was revived around 2006 with researchers able to train very deep neural networks by pre-training one hidden layer at a time using the unsupervised learning procedure [14]. Around the same time, availability of parallel graphics processors as commodity hardware and massive amounts of labeled data additionally propelled research in this area.

The organization of this report is as follows: In section 3, we review some of the important work that incorporated metadata to recommender systems. Section

4 will discuss the Movielens dataset and our approach of feature engineering and modeling using deep neural networks followed by experimental results in Section 5. Finally, we summarize the project in Section 6 and provide discussion for possible future work in Section 7.

3 Literature Review

We first review some of the important work that has been done with regards to extending recommendation techniques beyond the canonical user-item and item-item similarity and matrix factorization approaches. Shie et al [35] present an excellent taxonomy and a detailed survey of these papers. We also we look at the recent works that employ ideas from deep learning to advance state of the art of recommender systems.

Neighborhood based CF approaches measure the similarity between users or items based on the User-Item (U-I) matrix. As such, metadata can be exploited for calculating or refining the similarities and thereby improving recommendations. Content Boosted Collaborative Filtering by Melvile et al. in 2002 [26], was one of the first works that proposed the use of metadata such as title and genre to predict missing values in the U-I matrix and then deploying user-based CF to generate personalized recommendations. Their approach not only performed better than pure content and pure collaborative approaches, but also a naive hybrid that took an average of these two.

There has been prior work where tags are used to encode connections between users and items. Tso-Sutter et al. proposed a similarity fusion strategy that calculates the user-user (and item- item) similarity based on both tags and ratings within a CF framework [40]. They show that the fusion approach performed better than the user based and item-based models. Tagommenders by Sen et al. [34] proposed a group of tag-based recommendation algorithms that utilized inferred preferences for tags to predict the users preferences for items. They show that while the tag based algorithms performed worse than traditional CF algorithms for prediction task, they performed better for recommendation task and the hybrid of tag-based and funk-SVD performed the best overall.

As with neighborhood models, collaborative filtering methods based on latent factor models have also lent from the benefits of metadata. Collective matrix factorization (CMF) by Singh and Gordon [36], simultaneously factorizes multiple related matrices, including the U-I matrix and matrices containing metadata. For example, in the context of movie recommendation, CMF can jointly factorize both the user-movie rating matrix and another matrix containing metadata such as movie-genre matrix. This leads to diminish the sparsity problem when considering just the U-I matrix, leading to more effective latent factors.

The timestamp of user ratings has also been exploited for modeling and predicting user preferences on items The idea is that a users recent ratings on movies are more relevant rather than the users from a year ago. Ding and Li [9] used a decay function for the ratings so that more influence was given by the more recent ratings. The idea was subsequently used in the winning solution for the Netflix competition [19] where latent factors of users and items are designed as decay functions of time.

Tensor Factorization (TF) is another important class of algorithms within the latent factor models of CF. A tensor is a multidimensional or multimode array. In other words, tensor is a generalization of the matrix concept to multiple dimensions. TF is used in scenarios in which the data take the form of [user, item, interaction context, rating] in contrast to MF where the data take the form of [user, item, rating]. Two of the most popular TF methods used in CF are the CANDECOMP/PARAFAC (CP) model and the Tucker model by Kolda and Bader [18]. Rendle and Schmidt [28] proposed a Pairwise Interaction Tensor Factorization (PITF) model that explicitly models the pairwise interactions between users, items and tags to improve the performance of tag recommendation.

In addition to other significant work on TF models that exploit tags, researchers have also used TF models for exploring other information sources that are associated with user-item relationships. Multiverse Recommendations by Karatzoglou et al. [16] used contextual features such as companion, date of the week, year, season for movie recommendations and found TF based model to outperform matrix factorization methods and also other state of the art context aware approaches. Xiong et al proposed Probabilistic tensor factorization (PTF) [43], which combines Probabilistic Matrix Factorization (PMF) and TF for incorporating time information and were able to learn global evolution of latent features.

The idea of approaching the problem of recommendation as a classification task was made popular by Billsus and Pazzani [3], where they perform SVD for feature extraction followed by learning the data using a feed forward neural network and backpropagation. This was a landmark paper which opened the gates for application of machine learning to the task of movie recommendation. The recent advancement of research in training neural networks learning features has resulted in more work in this direction.

Salakhutdinov et al. [30] used Restricted Boltzmann machines, which are shallow, two-layer neural networks, for collaborative filtering. Along with presenting procedures to train and get predictions from these models, they showed that RBMs are able to outperform SVD on the Netflix dataset. Further, they demonstrated that a linear combination of predictions from SVD and RBMs improved error rate on the Netflix system by over 6 percent.

Strub et al [39] feed the sparse movie ratings to a SDAE with the aim of reconstructing dense rating vectors. The networks are trained to reconstruct

their inputs through a dimension reduction by performing a Non Linear Principal Component Analysis. Experiments on the Movielens and Jester datasets showed that autoencoders yielded results comparable to state of the art. Further, they showed that increasing the number of layers of autoencoders resulted in better performance.

Collaborative Deep Learning [41] combined deep learning for effective feature representation with collaborative filtering, to capture the implicit relationship between items. Using a tightly coupled model which on one hand uses content information for predicting ratings and on the other hand uses ratings for guiding effective learning of features from content, they were able to establish a general framework that outperforms the state of the art.

YouTube uses a feed forward neural network for candidate generation and subsequent fine-tuning to recommend a handful out of a corpus of million videos [8]. In addition to explicit rating information such as a thumbs up/down, they also use information such as time spent on a video by a user and age of a video to recommend content. They show that increasing the width of hidden layers improves results, as does increasing their depth. Their implementation of deep learning collaborative filtering model outperformed the matrix factorization approaches used previously at YouTube.

Deep learning had also been used to provide music recommendation and genre prediction. Wang et al. combine deep belief nets with CF to get a hybrid model that performs well in both warm start and cold start problems [42]. Hamel et al also used deep belief nets for automatic extraction of relevant features form musical audio for the purpose of genre prediction [11]. They found that this approach performs better than the traditional signal processing approach that involves extracting MFCC for learning features.

4 Methodology

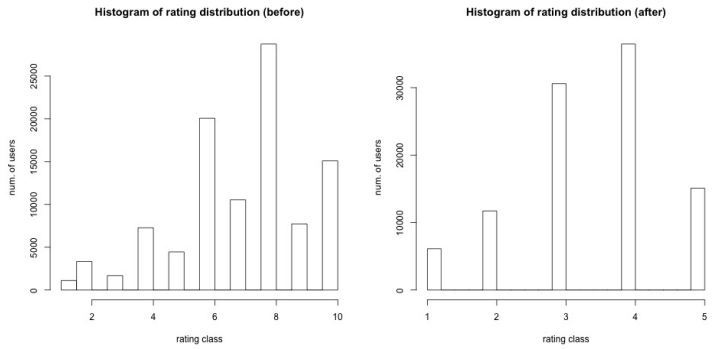
4.1 Dataset

We use the ml-latest-small[2] dataset available from the Movielens website. This contains movie ratings on the scale of 0.5 to 5 in increments of 0.5 and tagging activity from Movielens. It has a total of 100004 ratings and 1296 user generated tags across 9125 movies. The reason we chose this is twofold: Firstly, the popular ml-100k[3] dataset used for benchmarks does not contain information about the tags which was needed as a feature for the experiments. Secondly, ml-20m[4] contains the tag information but is quite large with 20M ratings and hence feature generation and model training would have been extemely slow.

[2] `http://grouplens.org/datasets/movielens/latest`
[3] `http://grouplens.org/datasets/movielens/100k`
[4] `http://grouplens.org/datasets/movielens/20m`

4.2 Problem Formulation

We treat the problem of movie recommendation as a multi-class classification problem. The original distribution of 10 classes, however is highly imbalanced with relatively very small number of observations for low ratings. To quantify, the imbalance ratio of highest to lowest majority classes is approximately 29:1. Hence, we formulate the problem into a 5-class classification problem formed as follows: a rating of 0.5 to 1.5 is treated as class 1, 2.0 and 2.5 as class 2, 3.0 and 3.5 as class 3, 4.0 and 4.5 as class 4 and 5.0 as class 5. By doing this, we are able to bring down the imbalance ratio to approximately 6:1. The following figure shows the distribution before and after the split.



4.3 Benchmark

As our goal is to ultimately provide good predictions for a user, we first benchmark our dataset. We use SVD, a well known CF algorithm based on latent factors. For our experiments, we used the SVD implementation in python-recsys[5], a python based library for building recommender engines.

We split the data into train (75%) and test (25%) and calculate MAE for different values of k (latent factors). We center the input data using $mean\_center = True$ and use default values for the remaining parameters while computing the SVD. The following table shows the mean result across 10 runs:

k	MAE
10	0.7075
20	0.7065
50	**0.7061**
100	0.7073

[5] `https://github.com/ocelma/python-recsys`

The best MAE of 0.7061 is obtained for $k = 50$. Hence, we use this value as the benchmark for our experiments in this project.

4.4 Features Used

Movieid and *userid*: These features, originally, are numeric and in themselves do not offer much for a model to learn. Both these features have been vectorized using an embedding layer. The embedding layer turns positive integers into dense vectors of fixed size. Here we use this layer layer to represent each movieid as a vector of size 32. The same was done with userid. The embedding method for intuition, can be thought of something similar to matrix decomposition [23].

Average Rating: This is the average rating of a given movie over all the ratings provided to it by different users. The is not a part of the ml-latest-small dataset and was available as `avg_rating` from the Movielens database. On inspection, this data was found to be sparse. Hence, we use the `imdb_id` (also available from the Movielens database) associated with each movie and retrieve the corresponding average ratings from the imdb website using an api[6]. Further, the `avg_rating` is normalized so that each entry is within the range [0,1]. For the small subset of movies for which the `avg_rating` was not available, it is set to zero.

Genre: There are 20 genres in our dataset so this feature is represented as a vector of size 20 where each genre is represented as an independent binary feature.

Tags: The set of total tags given by a user to a corresponding movie in the dataset accounts for 582 unique tags. These are also used as binary features similar to genres.

Rating: This is the target output of our model. It is a vector of 5 elements one for each of the 1-5 ratings.

We combine these features to create four different models described next.

4.5 Models

Firstly, we create a model `user_item`, contains the vectorized form of only userid and itemid. As a userid and itemid are now each vectors of size 32, the merging results into a combined dimension size of 64. This is our base model, which does not contain any metadata.

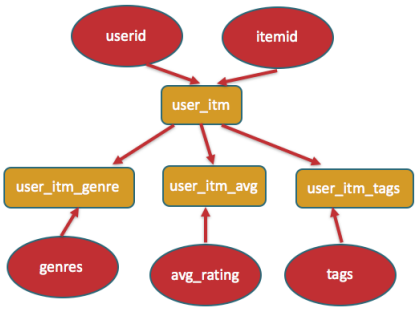
Next, we add the average rating to the `usr_item` model to create the `user_item_avg`. Since rating is just a numeric value, this makes the input dimension of size 65.

[6] `http://www.omdbapi.com`

Our third model is formed by adding the feature vector of genres to the `user_item` model. This results in the combined input vector of size 84.

Lastly, we create `user_item_tags`, a model formed by adding the tags information to the `user_item` model for a combined input vector of size 646.

The model formation from different features can be visualized using the following figure:



4.6 Network Training

We train a fully-connected, multi-layer neural networks on the four models. Each network has one input layer, one output layer and one or more hidden layers. In this section, we discuss the architecture and training of these networks.

4.6.1 Input Layer

The number of units in the input layer is completely and uniquely determined by the shape of the training data i.e. it is the same as the number of attributes within each sample of training data. However, we do not feed the entire feature vector to the network. Rather we make use of dropouts which is a simple and efficient way to avoid overfitting [38]. We use the residual values after the dropout as input to the network. Hence, the number of neurons in the input layer is the number of features that remain after dropout is applied to the initial model.

Due to the probabilistic nature of dropouts, we are essentially dropping different sets of neurons while training each sample. Heuristically, it can be thought of as training different neural networks. Hence, the dropout procedure is like averaging the effects of a very large number of different networks which will individually overfit in different ways, with the net effect being overall reduction in overfitting [27]. Thus, we end up with a network which learns more robust feature and perform better on the test set.

4.6.2 Hidden layers

The number of hidden layers and the number of hidden units within each layer are two important parameters in a neural network and are akin to selecting the best model from a set of algorithms. The selection of an optimal combination however is as much as art as it is a science.

4.6.2.1 Number of layers

Inspired by recent success in training deep neural networks in the field of computer vision, and from authoritative opinion[7], our approach for estimating the number of hidden layers is to start with a simple network with a single hidden layer and increase model complexity by adding one hidden layer at a time till the generalization error ceases to improve. We also use dropouts in after each hidden layer. In addition to this, we also use batch normalization [15] which we apply between the linearity and non-linearity within each hidden layer.

4.6.2.2 Number of nodes

The number of hidden units in a layer, again does not have a restriction albeit for the computational complexity. However, since we use regularizations such as dropouts and early stopping (explained in section 4.6.6), it is important to choose number of hidden units sufficiently large enough to prevent underfitting. It has been observed that using the same size for all hidden layers worked at least as good as the decreasing or increasing size. Also, an overcomplete[8] first hidden layer has been found to work better than an undercomplete one [2]. Hence, the hidden layers we use are identical for a given model. In our case, we found that the number of nodes in the hidden layers as 1.5 times the nodes in the input layer worked well.

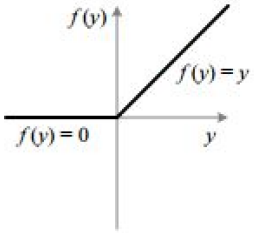
4.6.2.3 Activation Function

The activation function used in each hidden layer is *rectified linear units (relu)*. We use relu as opposed to other functions such as sigmoid or tanh as it converges faster due to less likelihood of the vanishing gradient [10] and provides better generalization [44].

$$\text{Relu: } f(x) = max(0, x)$$

[7] `https://www.quora.com/Artificial-Neural-Networks/Artificial-Neural-Networks-How-can-I-estimate-the-number-of-neurons-and-layers/answer/Yoshua-Bengio`

[8] greater than the input vector



4.6.3 Output Layer

In the output layer we use the softmax as the activation function and hence the number of nodes in this layer is the number of class labels which is five ratings on the scale of 1 to 5 stars. The output at the node j is then is given by the following equation:

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{5} e^{z_k}} \qquad \text{where } \mathbf{z} = \mathbf{wx}+\text{b}$$

The value of the above expression for all output j will be positive. In addition, the normalizing effect of denominator causes the output of the softmax to be like a probability distribution. We can then assign the class label by looking at the node which gets activated with the highest probability. This makes softmax a favorable function to use for the output. One can argue that we could have used a max-layer output which simply outputs a probability of 1 for maximum output of the previous layer. However such a function is not differentiable and hence difficult to train. On the other hand, a softmax works like max layer but is also differentiable and can be trained using gradient descent.

4.6.4 Loss Function

We use cross-entropy loss function given by the following equation:

$$C = -\frac{1}{n} \sum_x \sum_j y_j \ln a_j + (1 - y_j) \ln (1 - a_j)$$

We can easily interpret the above equation as a loss function. Firstly, it is always non-negative for all values of a. Secondly, we can see that the function approaches zero whenever the output value is close to the actual target y in both the cases of $a \approx 0$ and $a \approx 1$. Additionally, it has the benefit that unlike squared error, it avoids the problem of learning slowing down [27].

In general, we found that the combination of softmax function with cross-entropy is a popular choice for an output layer in deep learning models that aim to solve a multi-class classification problem.

4.6.5 Algorithm

We use ADAM [17] as the optimisation algorithm to train the network. It is an optimized implementation of the SGD that leverages the concept of momentum to do updates to the wieghts in a manner that leads to faster convergence. The idea is that parameters for which the direction of update changes frequently have updates of lower magnitude whereas those with constant direction of updates over multiple consecutive steps build up momentum and can afford to make larger updates and hence faster convergence. Moreover, ADAM has been known to slightly outperform other comparable algorithms, most notably RMSprop towards the end of optimization as gradients become sparser [17].

4.6.6 Number of Iterations (Epochs)

According to Benjio [2], this is a hyper-parameter that can be optimized almost for free with the use of early stopping. Early stopping is an inexpensive way to avoid overfitting to the training data by deciding how long should the training take place based on the error observed on unseen data e.g. the validation set as training progresses. As such, we use this in our network and thereby also obviate the need to decide a fixed number of epochs for training. In our experiments, we keep on training the model until the loss observed on validation set does not improve for more than 5 successive epochs.

The following table summarizes the overall network structure with number of nodes in each layer of each model:

	user_item (m1)	user_item_avg (m2)	user_item_genre (m3)	user_item_tags (m4)
Input layer (linear)	64	65	84	646
Each hidden layer (relu)	96	96	128	969
Output layer (softmax)	5	5	5	5

5 Experimental Results

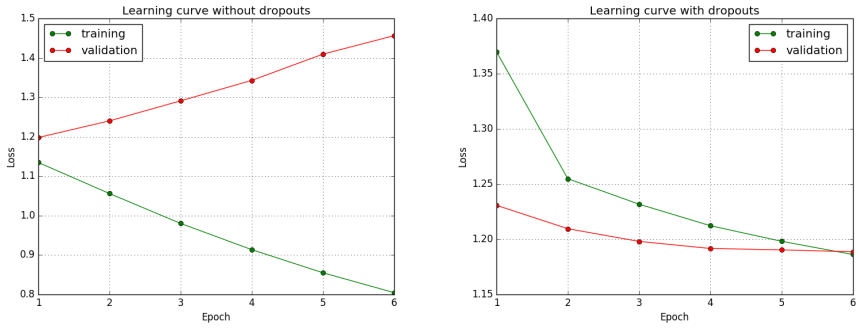
Since the Movielens system predicts and displays ratings that users would give to item, we evaluate our models based on predictive accuracy measures instead of its ability to recommend top items. This is because predictive accuracy can be used to recommend top items to the user; whereas, if we only evaluate top recommendations without good predictive accuracy, the displayed ratings would cause invalidation of the system [13]. Moreover, we do not use classification measures such as f1 scores as a rating of 4 misclassified as 3.5 is clearly better than the same classified as a 2 i.e. we must penalize the latter more than the former. MAE and RMSE are apt for this purpose as both are based on the

difference between the actual and predicted ratings. However, RMSE squares the error before aggregating them making it sensitive to outliers. Hence we use MAE, which for our purpose, is more robust of the two and also easier to interpret.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where $y_i \equiv$ Actual and $\hat{y}_i \equiv$ predicted

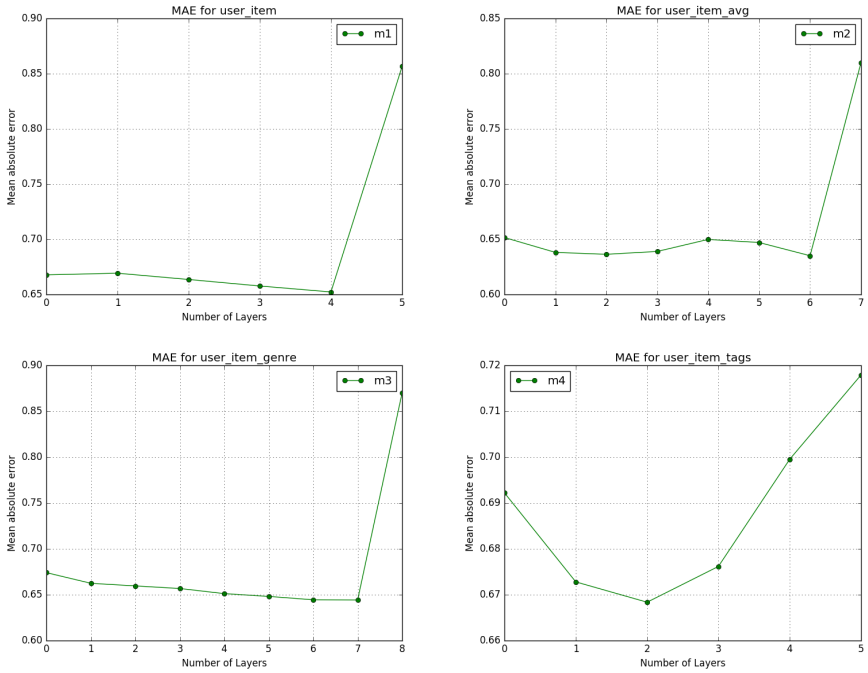
We split the data into training (75%) and validation (25%) sets and evaluate the performance of our model for different number of hidden layers. As a preliminary experiment, we run a neural network with a single hidden layer. However, we observe that this tends to overfit the data. This is evident from the large gap between the training and validation loss in the learning curve on the left in the diagram below for `user_item_avg` model.



We see that with each subsequent epoch, the training loss decreases whereas the validation loss increases. Hence, as a countering measure, we experiment with different amounts of dropouts between 0.1-0.8 as suggested by Hinton et al. [38] and find that dropouts of 0.6 to gives us a model the generalizes reasonably well. In addition, as discussed before, we also use early stopping as a method of regularization. The reduced overfitting is evident from the learning curve shown in the right box.

Hence, we select 0.6 in all the places where we use dropouts. Note that it is not because that it is the optimal value but because it is a reasonable value. We say this because, different layers within the models can each have different values of dropout and tuning this across all the four models can take a significant amount of time.

Next we experiment with increasingly complex models by adding one hidden layer at a time and record the best MAE obtained on each of the four models.



In general we see lower MAE values when networks have hidden layers compared to when they do not. This tells us that adding non-linearity to the network enables it to capture patterns in the dataset that the linear model is unable to. When experimenting with different number of layers for each model separately, we see a large value of MAE after reaching a certain point as evident from the spike in each of them. This tells us that beyond a certain level of complexity, the network is simply able memorize the data. Hence we stop adding layers further.

Dataset	**MAE**
SVD	0.7061
user_item	0.6524
user_item_avg	0.6352
user_genre	0.6443
user_tags	0.6684

The above table summarizes the best error rates obtained for each model. In general, we see that all the four models perform better than the benchmark of SVD that we obtained on the same dataset. In particular, the `user_item_avg` and `user_item_genre` give us improvement in MAE over our base model `user_item`. This tells us that deep neural networks when used as an instrument not only can provide good results for movie recommendation, but when provided with

additional metadata can learn from that to further improve the quality of predictions. We also note that `user_item_tags` performs worse than our base model. We attribute this to the high sparsity of this model which could have led the network to treat it as noise instead of useful information.

6 Conclusion

In this project, we have explored the application of deep neural networks to the problem of movie recommendations. We treat the prediction of star ratings as a multi-class classification problem with 5 classes and use MAE to evaluate the predicted results. We construct four models containing a combination of movieid, userid, average rating, tags and genres, where we use vector embedding to represent movieid and userid. We train deep neural networks with different number of hidden layers on each of the four models and record the minimum MAE obtained. We show that the base model containing only userid and movieid trained on our network yielded a significantly better MAE than SVD on the original dataset. Finally, we show that the networks are able to learn from metadata, specifically average rating and genre which yield better MAE than the base model without metadata.

7 Discussion and Future Work

For a long time, matrix factorization methods were the best performing for recommendation problems. These methods work by learning latent factors within the data. However, MF approaches like SVD can only represent linear combinations of these features. As discussed in section 2.4, the strength of neural network comes from being able to learn non-linear representations in the data. This is made possible by the use of non-linear activation functions in the hidden layers. Each node within the hidden layers detects a feature- a template. When the pattern of information it is receiving from the preceding layer matches that template, it becomes activated, and sends signals of its own to the next layer. Hence, when we add hidden layers into a neural network, we are essentially taking information from the previous layer and transforming into a progressively complex, higher-level representation- Complex because it can now identify a non-linear combination of features identified in the previous layer; higher-level means that it now contains a compact and more salient representation of that data. This notion of transforming data into smaller but more meaningful information is the primary advantage of neural networks over linear methods.

The architecture of neural network can greatly affect the overall prediction outcome for a model. While greater number of hidden layers can help the network learn more complex representations from the data, it can also make the training increasingly difficult. On the other hand, too large or too small number of neurons in the hidden unit casues problems with overfitting and under fitting re-

spectively. While it is difficult to precisely quantify these measures for a perfect model, extensive research has led to practical recommendations [2] regarding these decisions which we have used as a guiding principle in our approach to designing the network. Ultimately, the selection of the optimal architecture comes down to trial and error.

The importance of average rating for predicting movies that we found in our experiments agrees with the basic premise of collaborative filtering which says that ratings given by a user to an item has a non trivial relation to the ratings given to the same item by other users. In fact, we can apply our learnings from collaboarative filtering and arrive at new ways of feature engineering to train machine learning models: Instead of looking at the overall average ratings for each item, we could consider having a weighted average which can be calculated by applying greater weights to other users in the neighborhood of the user under consideration. In addition, we also find worth exploring the effect of reducing the user to user variablility in ratings on the overall prediction accuracy. This can be done by using the z-score instead of the actual star rating given by a user as can readily compute the mean and variance of all the ratings given by a user from our data.

A limitation of this work is that given the number of distinct tags present, there can be better ways of incorporating tags into the model than one-hot encoding. e.g. sci fi should obviously be treated as the same tag as sci-fi; family, happiness, heartwarming may be treated as similar to each other and so can creative, imaginative and surreal. An approach of clustering followed by a dimensionality reduction using embedding might be appropriate. We consider this idea as a potential future work. Additionally, one could argue that perhaps a comparison of neural networks with a non-linear approaches such as Non Linear Probabilistic Matrix Factorization (NL-PMF) [21] would be more appropriate. While this would be interesting to study, we found SVD being more commonly used was suitable for our purpose of a benchmark algorithm.

We began our initial implementation in Torch which is a machine learning and scientific computing framework based on the Lua programming language. However, we found out that a familar programming ecosystem that contains simpler abstractions for complex algorithms was crucial given the nature of the project. We later considered the Keras library which is based on Python and found it an excellent framework for this purpose with better documentation and code examples. Moreover, as it supports the use of TensorFlow in addition to Theano and as backend, one can expect to benefit from massive engineering efforts that Google is putting into making TensorFlow a faster and more scalable deep learning engine in the future.

References

[1] Basu, C., Hirsh, H., Cohen, W., et al. Recommendation as classification: Using social and content-based information in recommendation. In *Aaai/iaai* (1998), pp. 714–720.

[2] Bengio, Y. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[3] Billsus, D., and Pazzani, M. J. Learning collaborative information filters. In *Icml* (1998), vol. 98, pp. 46–54.

[4] Billsus, D., and Pazzani, M. J. User modeling for adaptive news access. *User modeling and user-adapted interaction 10*, 2-3 (2000), 147–180.

[5] Burke, R. Integrating knowledge-based and collaborative-filtering recommender systems. In *Proceedings of the Workshop on AI and Electronic Commerce* (1999), pp. 69–72.

[6] Burke, R. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction 12*, 4 (2002), 331–370.

[7] Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., and Sartin, M. Combining content-based and collaborative filters in an online newspaper. In *Proceedings of ACM SIGIR workshop on recommender systems* (1999), vol. 60.

[8] Covington, P., Adams, J., and Sargin, E. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems* (2016), ACM, pp. 191–198.

[9] Ding, Y., and Li, X. Time weight collaborative filtering. In *Proceedings of the 14th ACM international conference on Information and knowledge management* (2005), ACM, pp. 485–492.

[10] Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In *Aistats* (2011), vol. 15, p. 275.

[11] Hamel, P., and Eck, D. Learning features from music audio with deep belief networks. In *ISMIR* (2010), vol. 10, Utrecht, The Netherlands, pp. 339–344.

[12] Herlocker, J. L., Konstan, J. A., and Riedl, J. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work* (2000), ACM, pp. 241–250.

[13] Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS) 22*, 1 (2004), 5–53.

[14] Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation 18*, 7 (2006), 1527–1554.

[15] Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[16] Karatzoglou, A., Amatriain, X., Baltrunas, L., and Oliver, N. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems* (2010), ACM, pp. 79–86.

[17] Kingma, D., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[18] Kolda, T. G., and Bader, B. W. Tensor decompositions and applications. *SIAM review 51*, 3 (2009), 455–500.

[19] Koren, Y. Collaborative filtering with temporal dynamics. *Communications of the ACM 53*, 4 (2010), 89–97.

[20] Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer 42*, 8 (2009).

[21] Lawrence, N. D., and Urtasun, R. Non-linear matrix factorization with gaussian processes. In *Proceedings of the 26th Annual International Conference on Machine Learning* (2009), ACM, pp. 601–608.

[22] Le, Q. V., et al. A tutorial on deep learning part 1: Nonlinear classifiers and the backpropagation algorithm, 2015.

[23] Levy, O., and Goldberg, Y. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems* (2014), pp. 2177–2185.

[24] Linden, G., Smith, B., and York, J. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing 7*, 1 (2003), 76–80.

[25] Lops, P., De Gemmis, M., and Semeraro, G. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*. Springer, 2011, pp. 73–105.

[26] Melville, P., Mooney, R. J., and Nagarajan, R. Content-boosted collaborative filtering for improved recommendations. In *Aaai/iaai* (2002), pp. 187–192.

[27] Nielsen, M. A. Neural networks and deep learning. *URL: http://neuralnetworksanddeeplearning. com/.(visited: 01.11. 2014)* (2015).

[28] Rendle, S., and Schmidt-Thieme, L. Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the third ACM international conference on Web search and data mining* (2010), ACM, pp. 81–90.

[29] Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review 65*, 6 (1958), 386.

[30] Salakhutdinov, R., Mnih, A., and Hinton, G. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning* (2007), ACM, pp. 791–798.

[31] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web* (2001), ACM, pp. 285–295.

[32] Sarwar, B. M., Konstan, J. A., Borchers, A., Herlocker, J., Miller, B., and Riedl, J. Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work* (1998), ACM, pp. 345–354.

[33] Schwab, I., Kobsa, A., and Koychev, I. Learning user interests through positive examples using content analysis and collaborative filtering. *Internal Memo, GMD, St. Augustin, Germany* (2001).

[34] Sen, S., Vig, J., and Riedl, J. Tagommenders: connecting users to items through tags. In *Proceedings of the 18th international conference on World wide web* (2009), ACM, pp. 671–680.

[35] Shi, Y., Larson, M., and Hanjalic, A. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys (CSUR) 47*, 1 (2014), 3.

[36] Singh, A. P., and Gordon, G. J. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (2008), ACM, pp. 650–658.

[37] Smyth, B., and Cotter, P. A personalised tv listings service for the digital tv age. *Knowledge-Based Systems 13*, 2 (2000), 53–59.

[38] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research 15*, 1 (2014), 1929–1958.

[39] Strub, F., and Mary, J. Collaborative filtering with stacked denoising autoencoders and sparse inputs. In *NIPS Workshop on Machine Learning for eCommerce* (2015).

[40] Tso-Sutter, K. H., Marinho, L. B., and Schmidt-Thieme, L. Tag-aware recommender systems by fusion of collaborative filtering algorithms. In *Proceedings of the 2008 ACM symposium on Applied computing* (2008), ACM, pp. 1995–1999.

[41] Wang, H., Wang, N., and Yeung, D.-Y. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2015), ACM, pp. 1235–1244.

[42] Wang, X., and Wang, Y. Improving content-based and hybrid music recommendation using deep learning. In *Proceedings of the 22nd ACM international conference on Multimedia* (2014), ACM, pp. 627–636.

[43] Xiong, L., Chen, X., Huang, T.-K., Schneider, J., and Carbonell, J. G. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the 2010 SIAM International Conference on Data Mining* (2010), SIAM, pp. 211–222.

[44] Zeiler, M. D., Ranzato, M., Monga, R., Mao, M., Yang, K., Le, Q. V., Nguyen, P., Senior, A., Vanhoucke, V., Dean, J., et al. On rectified linear units for speech processing. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on* (2013), IEEE, pp. 3517–3521.