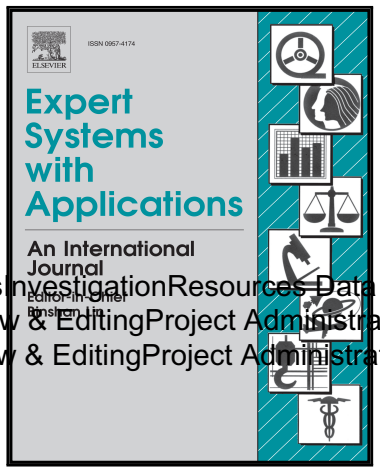
Journal Pre-proof



DNNRec: A Novel Deep Learning based Hybrid Recommender System

Kiran R ConceptualizationMethodologySoftwareValidationFormal AnalysisInvestigationResources Data CurationW
Pradeep Kumar ValidationInvestigationWriting-Original DraftWriting-Review & EditingProject Administration ,
Bharat Bhasker ValidationInvestigationWriting-Original DraftWriting-Review & EditingProject Administration

PII: S0957-4174(19)30771-7
DOI: https://doi.org/10.1016/j.eswa.2019.113054
Reference: ESWA 113054

To appear in: *Expert Systems With Applications*

Received date: 17 December 2018
Revised date: 18 August 2019
Accepted date: 25 October 2019

Please cite this article as: Kiran R ConceptualizationMethodologySoftwareValidationFormal AnalysisInvestigationRe
Pradeep Kumar ValidationInvestigationWriting-Original DraftWriting-Review & EditingProject Administration ,
Bharat Bhasker ValidationInvestigationWriting-Original DraftWriting-Review & EditingProject Administration ,
DNNRec: A Novel Deep Learning based Hybrid Recommender System, *Expert Systems With Applications* (2019), doi: https://doi.org/10.1016/j.eswa.2019.113054

This is a PDF file of an article that has undergone enhancements after acceptance, such as the addition of a cover page and metadata, and formatting for readability, but it is not yet the definitive version of record. This version will undergo additional copyediting, typesetting and review before it is published in its final form, but we are providing this version to give early visibility of the article. Please note that, during the production process, errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

© 2019 Published by Elsevier Ltd.

Highlights

- A novel hybrid Deep Learning Based Recommender System 'DNNRec' is proposed
- DNNRec leverages embeddings, combines side information and a very deep network
- DNNRec addresses cold start case and learns of non-linear latent factors
- Proposed solution is benchmarked against existing methods on accuracy and run time
- DNNRec outperforms state-of-the-art methods overall and in cold start case

Journal Pre-proof

Journal Pre-proof

DNNRec: A Novel Deep Learning based Hybrid Recommender System

Kiran R, Pradeep Kumar, Bharat Bhasker

(efpm04013@iiml.ac.in, pradeepkumar@iiml.ac.in, bhasker@iiml.ac.in)

Information Technology and Systems

Indian Institute of Management Lucknow, India

*: *Corresponding Author*[1]

Abstract

We propose a novel deep learning hybrid recommender system to address the gaps in Collaborative Filtering systems and achieve the state-of-the-art predictive accuracy using deep learning. While collaborative filtering systems are popular with many state-of-the-art achievements in recommender systems, they suffer from the cold start problem, when there is no history about the users and items. Further, the latent factors learned by these methods are linear in nature. To address these gaps, we describe a novel hybrid recommender system using deep learning. The solution uses embeddings for representing users and items to learn non-linear latent factors. The solution alleviates the cold start problem by integrating side information about users and items into a very deep neural network. The proposed solution uses a decreasing learning rate in conjunction with increasing weight decay, the values cyclically varied across epochs to further improve accuracy. The proposed solution is benchmarked against existing methods on both predictive accuracy and running time. Predictive Accuracy is measured by Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE) and R-squared. Running time is measured by the mean and standard deviation across seven runs. Comprehensive experiments are conducted on several datasets such as the MovieLens 100K, FilmTrust, Book-Crossing and MovieLens 1M. The results show that the proposed technique outperforms existing methods in both non-cold start and cold start cases. The proposed solution framework is generic from the outperformance on four different datasets and can be leveraged for other ratings prediction datasets in recommender systems.

Keywords:

Deep Learning; Recommender Systems; Embeddings; Side Information; Cyclical Learning Rates; Deep Neural Network; Cold Start problem

[1] Corresponding Author Address: Indian Institute of Management Lucknow, IIM Rd, Prabandh Nagar, Mubarakpur, Lucknow, Uttar Pradesh 226013; Phone: +91-9900601013

Journal Pre-proof

1 Introduction

Recommender systems are increasingly used for suggesting movies, music, videos, e-commerce products or other items. They accelerate the process of search for the users and help businesses maximize their sales (Lee, Huntsman, Fl, Huntsman, & Fl, 2014). Given the research focus on recommender systems and the business benefits of higher predictive accuracy of recommender systems, there has been an increasing focus on building better solutions. Recommender systems predict the future preference for a set of items for a user either as a rating or as a binary score or as a ranked list of items. Popular recommender systems like the MovieLens recommender system, Amazon and Netflix express the user preference as a numeric rating. When the rating is numeric, one of the metrics like Root Mean Squared Error (RMSE), Mean Average Error (MAE), Mean Squared Error (MSE) or R-squared are used to measure the predictive accuracy of the recommender system. When the problem is one of ranking, metrics like NDCG (Normalized Discounted Cumulative Gain) are used. The scope of this paper is the ratings prediction case of recommender systems. The goal is one of making the predictions as accurate as possible. Along-with predictive accuracy, the other challenges in recommender systems include cold start (Lam, Vu, & Le, 2008), sparsity and candidate generation (Covington, Adams, & Sargin, 2016). One of the important problems in recommender systems is the cold start problem. Cold Start Problem is one of "Making recommendations where there are no prior interactions available for a user or an item". The cold start problem has been classified into user cold start and item cold start, which refer to cases of insufficient examples of items and users respectively (Bernardi, Kamps, Kiseleva, & Mueller, 2015). Over the last few years, several techniques have been developed and benchmarked on publicly available datasets. There have been several approaches in literature to solve the recommender system problem. These approaches have been classified into Content-based (CB), Collaborative Filtering (CF) based and Hybrid methods. Popular buckets include Heuristic methods, Matrix Factorization based Collaborative Filtering methods, Neighborhood based Collaborative Filtering methods and Machine Learning methods. To address the cold start problem, CB features extracted for users and items are used as inputs in several machine learning techniques like regularized regression and tree-based methods. Given the strategic importance of recommender systems both in research and business and the algorithmic advancements, improving predictive accuracy is an important task. Netflix released more than 100 million customer generated movie ratings as part of the Netflix Prize competition with a goal to reduce the RMSE of the test data by 10 percent, with a prize of USD 1 million to the first team to reach that goal (Netflix, 2009).

Deep Learning methods have seen huge success in computer vision beating the state-of-the-art results in many computer vision applications (Lecun, Bengio, & Hinton, 2015). Deep Learning Methods are a branch of machine learning that build neural networks with complicated architectures (like multiple

hidden layers, or loops, or convolutions with shared weights and pooling layers). The building blocks of Deep Learning are Neural networks, which are infinitely flexible functions. Deep Learning methods also have an ability to do all-purpose parameter fitting through advanced stochastic gradient descent algorithms, while achieving speed and scalability using Graphical Processing Units (GPU). The last few years have seen the increasing performance and availability of GPUs for parallel matrix computations, that are at the core of deep learning. The availability of frameworks like Pytorch (Facebook), TensorFlow (Google) and MXnet (Apache) have accelerated the use of deep learning methods. There are different types of deep learning methods like Deep Neural Networks (DNN), Convolutional Neural Networks (CNN), Recurrent Neural Network (RNN) types like Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks, unsupervised methods like Autoencoders and Generative Adversarial Networks or GANs (Lecun et al., 2015). All these have enabled rapid adoption and research in deep learning systems. Deep Learning systems are now increasingly being applied to structured datasets and form an area of active research. There have been several advances in Deep Learning on speeding up running time by effectively finding the optimal learning rates (Smith, 2017a), learning complex features from the data with minimum external contribution (Araque, Corcuera-Platas, Sánchez-Rada, & Iglesias, 2017) and modern stochastic gradient interventions that can find the minima more efficiently to deliver state-of-the-art results.

In this light, the motivation for this study is to seek answers to the following questions:

1. Can the predictive accuracy in comparison to the existing CF techniques be improved by the non-linear latent features engineered from the embeddings?
2. Can the performance in the cold-start case benefit from the side information about the users and items that are incorporated into the deep neural network?
3. Can the proposed deep neural network work across different datasets for ratings prediction so that it can be a generic approach to solve recommender system problems in the ratings prediction case?
4. Given that learning rate and weight decay greatly influence the convergence of a DNN, how can they be optimized for highest predictive accuracy?
5. What is the tradeoff in running time of deep learning-based recommender systems vs. existing methods?

In this paper, a new deep learning-based hybrid recommender system is proposed. The solution makes four major contributions. First, it alleviates the cold start problem by utilizing side information about users and items into a DNN, where-ever such auxiliary information is available. Second, it retains the benefits of traditional matrix factorization techniques by learning latent factors and builds on them to

Journal Pre-proof

learn non-linear latent factors as well using embeddings. Third, it enables achieving highest predictive accuracy by using cyclical learning rates and decaying weights across epochs. This variation ensures that the solution does not get stuck in a local minima as explained in the solution section. The fourth contribution is showing that the solution is viable from both optimizing criteria (RMSE, R-squared, MSE and MAE) and satisficing criteria (mean and standard deviation of running times across seven runs). The performance is measured in both in overall and cold start cases on four datasets namely MovieLens 100K, FilmTrust, Book-Crossing and MovieLens 1M. Experimental results show that the proposed solution beats the existing benchmarks and alleviates the cold start problem. Recommender systems are an intelligent system that find application not only in e-commerce but in several industries. Our paper presents an approach to address the gaps in collaborative filtering approach regarding non-linear latent factors and performance in cold start cases. We design a recommender system to address the gaps in CF approaches and test it on five different datasets improving performance. We address different facets in building a more accurate recommender system solution using deep learning.

The remainder of this paper is organized as follows. Section 2 provides a taxonomy of related recommender system methods that are subsequently benchmarked against. Section 2 also describes related work on recommender systems and usage of deep learning on structured data and recommender systems. Section 3 outlines the proposed deep learning solution DNNRec and its theoretical framework, describing the key components of the same. Section 4 details the experimental setup comprising of the hardware, software packages, datasets and validation set creation. Section 5 summarizes the experimental results on four different datasets namely MovieLens 100K, FilmTrust, MovieLens 1M and Book-Crossing. Section 5 also highlights the superiority of the proposed solution against existing methods. Finally, Section 6 outlines the conclusions from the work and opportunities for future research.

2 Background and Related Work

The dominant approaches in recommender systems are broadly classified into three categories namely Content Based (CB), Collaborative Filtering (CF) and Hybrid methods. CB methods utilize the content of the items to create features to match user profiles. These methods leverage user features and item features to predict the ratings using several machine learning methods. CB methods could include Context based Recommender systems (Aggarwal, 2016) that use space and time features of the users and items as well. On the other hand, CF methods use the relationship between users and items that are encoded in the ratings matrix to make predictions. CF methods leverage similarities between users and items to predict the ratings. CF methods are classified into nearest neighbor CF and matrix factorization CF methods. Nearest neighbor methods provide the average rating of the “k” nearest neighbors. When these neighbors are users, it is called user-KNN and when the neighbors are items, it is called item-KNN

(Jannach, Lerche, Kamehkhosh, & Jugovac, 2015). Matrix Factorization methods create a profile for each user and each item, based on previous user-item interactions like historical ratings. Neighborhood methods are centered on evaluating a user's preference based on 'nearby' users or items, while matrix factorization methods are latent factor models that characterize both user and item profiles at the same time (Koren, Bell, & Volinsky, 2009). Singular Vector decomposition (SVD) is a popular matrix factorization technique that provides a lower rank approximation of the ratings matrix and is used in recommender systems to capture the latent relationships between users and items and also to produce a low-dimensional representation of the original user-item space. This low-dimensional representation is then used to compute the neighborhood in the reduced space (Sarwar, Karypis, Konstan, & Riedl, 2000). SVD was also used in the winning entry for the Netflix Prize (Gomez-Uribe & Hunt, 2015; Netflix, 2009). SVD++ improves SVD by adding biases or intercepts that are independent of any interactions. In SVD++, the full rating value is not just an interaction between the users and items, but involves a combination of that interaction with portions that individual user or item biases explain (Koren et al., 2009). The latent factors for each user and item can be learnt using gradient descent with an objective to get the best accuracy for the predicted ratings against the actual ratings for each user-item pair (Koren, 2008). There are variants where deep learning has been employed on top of the latent factors of users and items, stacking them into a single vector of inputs and having multiple hidden layers before a final output layer (Zhang, Chen, Wang, & Shi, 2018). In another study, deep learning is employed in CF to solve the cold start problem (Wei, He, Chen, Zhou, & Tang, 2017). CB methods include heuristic methods and machine learning methods that leverage the user and item content-based features to make predictions. Heuristic methods for Recommender Systems predict the overall average rating or the average rating for the user or the average rating for the item and these are referred to as the Global Average, Global User Average and Global Item Average methods respectively (Adomavicius & Tuzhilin, 2005). CB methods add features for each user and item and then use them in traditional machine learning models (Lops, de Gemmis, & Semeraro, 2011). Hybrid methods combine CB and CF to make predictions. The challenges in recommender systems can be classified into those related to predictive accuracy like RMSE, MSE MAE, R-squared and those related to satisficing criteria like scalability and speed, those related to candidate generation and those related to the nature of the data like cold start, sparsity and imbalance.

Various deep learning methods have been applied to recommender systems like Restricted Boltzmann Machines (RBM) per user approach (Salakhutdinov, Mnih, & Hinton, 2007), Neural Collaborative Filtering (He et al., 2017), Deep Collaborative Neural Networks (Catherine & Cohen, 2017) among others. Attempts have been made to solve the cold start problem using deep learning. These methods include extracting features from images using convolutional neural networks (Oord, Dieleman, & Schrauwen, 2013), extracting audio based features using LSTM-based recurrent neural networks

Journal Pre-proof

(Balakrishnan, 2014) etc. Converting item text into latent features (Bansal, Belanger, & McCallum, 2016) has been used in the case of textual data as well. The proposed solution seeks to integrate the side information about the users and items present in the datasets into a deep neural network in combination with user and item embeddings. Another approach in deep learning literature has been using autoencoders in a specific deep neural network called SDAE in combination with a CF model, TimeSVD++ (Wei et al., 2017). In another study, a Deep Neural Network approach with multiple hidden layers of linear and non-linear activations was employed and compared to a classic machine learning approach for Natural Language Processing (Do, Prasad, Maag, & Alsadoon, 2019). Stacked autoencoder based CF is one of the deep learning approaches. An example of such a novel technique is called AutoRec (Sedhain et al., 2015a). There has been an attempt to refine the training approach of autoencoders to integrate side information (Strub et al., 2016) by extending the construction of embeddings by neural networks to a collaborative filtering setting. COFILS, Collaborative Filtering to Supervised Learning (Braida et al., 2015) approach was proposed to transform the Collaborative filtering dataset into a supervised learning dataset by using the latent features of the CF as features in a supervised learning model. Deep Collaborative Filtering (DCF)(Li et al., 2015a) was proposed to make use of the side information present in recommender datasets about users and items.

3 DNNRec: A hybrid Deep Learning-based Recommender System

In this section, the proposed DNNRec recommender system algorithm is described in detail. A recommender system is designed for generating recommended rating for each user and item pair. Consider U users of which there are U_{CS} cold start users ($U_{CS} \subset U$) and I items, of which there are I_{CS} cold start items ($I_{CS} \subset I$), Let N be the number of ratings. Rating $r_{i,j,t}$ denotes the rating by user i on item j at time t. Table 1 presents the list of abbreviations and notations used in this paper with their meanings. The recommender systems task considered in this paper is to predict the unknown ratings given the known ratings.

Table 1. List of Notations

Symbol	**Meaning**
U	Total Number of Users
$U_{CS} \subset U$	Total Number of Cold Start Users
N_U	Number of Users
K_U	Number of User Factors
I	Total Number of Items
$I_{CS} \subset I$	Total Number of Cold Start Items
N_I	Number of Items

K_I	Number of Item Factors
N	Number of Ratings
$r_{i,j,t}$	Rating to item j by user i at time t
$E_{feature}$	Embedding matrix for any feature, E_{user} is the embedding matrix for a user and E_{item} is the embedding matrix for an item
V_x	One column of $E_{feature}$ (i.e. an embedding vector for a particular user or item value) corresponding to input x
K	Number of Embeddings
x	One-hot encoded representation of categorical variable x
l	Number of levels in the categorical feature
W_H	Weight matrix at hidden layer H. Used to apply an affine transformation to the inputs of the prior layer namely the matrix product WX
b_H	Bias of the hidden layer H
X_H	Output of hidden layer H
h_i	Number of hidden nodes in hidden layer i
p	Dropout Probability
D	Number of dense features in the side information related to the users or items
C	Number of categorical features in the side information related to the users or items
K_C	Number of Embeddings for the categorical features
X	Input identifier
k	Number of unique levels in the embedding
$LeakyReLU(x)$	LeakyReLU(x)=max(x,0.01)
$ReLU(x)$	ReLU(x)=max(0,x)
$dropout$	Dropout layer where every activation is randomly turned off with a probability of p. Dropout serves as a regularization method
$concat(v_1, v_2)$	Combine vectors v_1 and v_2 into a single vector
T_{ij}	matrix with a cell value t_{ij} = 1 only if the user i rated item j and 0 otherwise
r_{ij}	denotes the rating provided by user i to item
$matmul$	Matrix Multiplication

The key components of the proposed solution are explained below.

3.1 Embedding for Categorical features

The traditional way to represent users and items, which are both categorical variables, is as dummy variables which are also known as one-hot encoded vectors. These vectors are very sparse and are represented using special sparse matrices. This approach suffers from the drawback of failing to capture semantic relationships between the users and items as it provides each user and item value, its own representation. The proposed solution uses an alternate representation called embeddings, which are representations of each of the categorical values as vectors in N-dimensional space. Embeddings are

Journal Pre-proof

lookup matrices that return an array of size K, where K is the number of embeddings. For each user identifier, an array of size K_U is returned and an array of size K_I is returned for each item identifier. This is illustrated in the Figure 1. This is easy to contrast against traditional methods where the user IDs would be encoded with a dummy variable. Embeddings allow the deep learning algorithm to learn a richer representation for the users and the items jointly. This is similar to the natural language modeling approaches where each word is mapped to a vector space of a fixed size (Goldberg et al., 2014). This approach has been successfully employed for YouTube recommendations (Covington et al., 2016) using deep neural networks, for Taxi Destination prediction using artificial neural networks (De Brébisson, Simon, Auvolat, Vincent, & Bengio, 2015), for approximating categorical variables into Euclidean spaces in the Rossman Forecasting competition and in word embeddings in Word2vec (Goldberg et al., 2014; Mikolov et al., 2013).

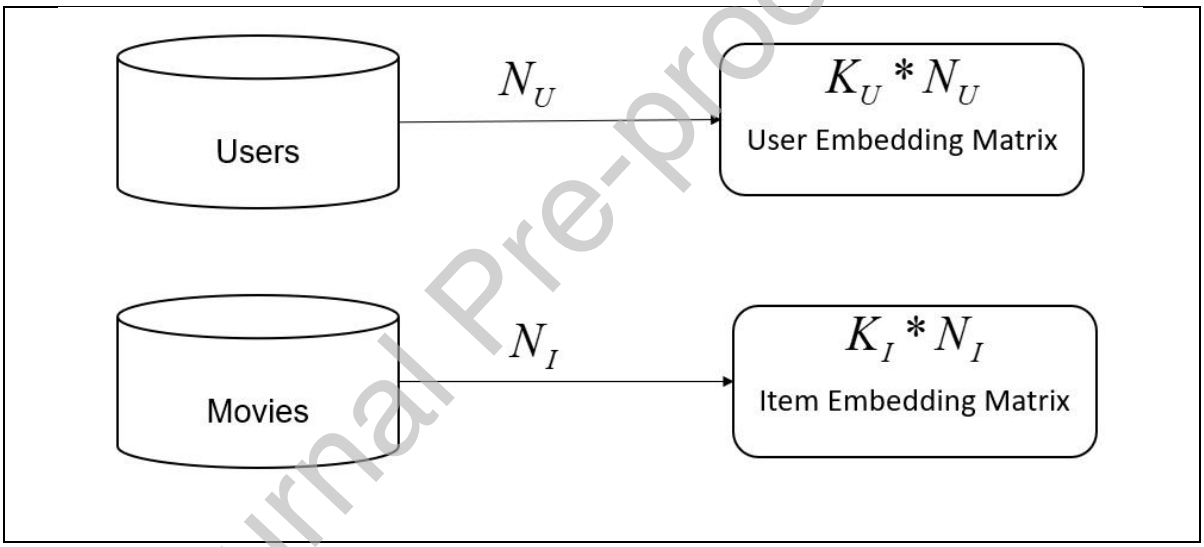


Figure 1. User and Movie Embeddings in for the MovieLens 20M dataset

Mathematically, for a categorical feature with l unique levels, the embedding of k factors is denoted by a matrix $E_{feature} \in \mathbb{R}^{l*k}$, in which each row $E_{feature}$ corresponds to the embeddings for a level of the categorical feature. So, the columns of $E_{feature}$ correspond to the different unique levels of the categorical feature and the rows of $E_{feature}$ correspond to the embeddings. A column of $E_{feature}$ corresponds to the embedding for an input level of the categorical variable. For a level x, V_x would denote the embedding matrix values. An input one-hot encoded vector x of size $1*k$ is transformed into its embedding by using the matrix-vector product as shown in equation (1) below.

$$V_x = x.matmul(E_{feature}) \quad (1)$$

There are two embedding matrices E_{user} for the users and E_{item} for the movies. The number of embeddings (also called number of factors) K_U and K_I are both hyper-parameters that are to be chosen by the modeler and tuned like other hyper-parameters in the network. Embeddings for users and items are like the latent factors that are used in collaborative filtering, with the difference being that the values of these embeddings are like any other parameters like the weight matrices at the different hidden layers that are learnt by the deep learning network. As a result, they can capture both linear and non-linear user and item factors. The weights of these embeddings are updated differently from collaborative filtering as the matrices of embeddings for the users and items are included as model parameters, randomly initialized and modified by stochastic gradient descent like the other parameters.

3.2 Hybrid Recommender System

The proposed solution combines the user and item embeddings as features with side information regarding users and items. The combination of the embeddings and the content-based features result in the proposed solution being a hybrid recommender system. The side information available in different datasets that were used for experimentation in this paper is denoted in Table 2.

Table 2. Side Information about Users and Items in the Different datasets

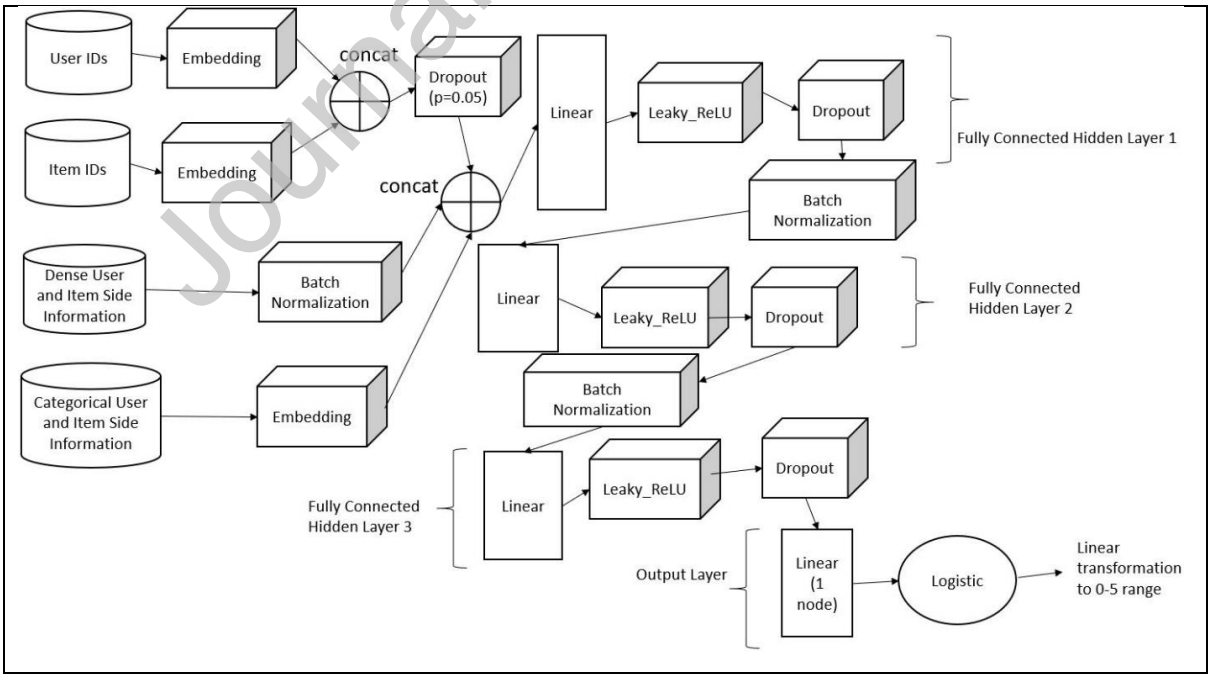
Dataset	**Side Information about Users**	**Side information about Items**
MovieLens 100K	Item	Tag, Title, Genre of Movies
	User	Not Available
FilmTrust	Item	Not Available
	User	Trust ratings on the User on other users and vice-versa
Book-Crossing	User	Age, Location, Author Name
	Item	Year of Publication. Publisher, Book Title
MovieLens 1M	User	Gender, Occupation, Zip code
	Item	Genre, Title

3.3 Deep Neural Network

The proposed architecture consists of three hidden layers. Based on computing capacity, this could be extended to more hidden layers. The number of hidden layers, the number of nodes in each hidden layer and the activation functions in each layer are all hyper-parameters that could be tuned by the modeler. Each hidden layer computes a linear function followed by a *LeakyReLU* (Lecun et al., 2015) function followed by a dropout. Each of the hidden layers employs batch normalization. The last layer is a fully

connected linear layer with 1 hidden node, that is input into a sigmoid activation function. The proposed solution controls overfitting by using algorithmic methods like dropout and regularization. The proposed solution creates embedding matrices of users and movies and stacks them along-with the side information for the users and items that are available in the datasets and described in Table 2. The proposed solution is flexible enough to allow for more types of dense or categorical features from the side information to be included. The proposed deep learning solution architecture is represented in Figure 2. As a result, if there are U number of users, I number of items, C number of categorical features and D number of dense features, K_U user embeddings and K_I movie embeddings, $K_U + K_I + K_C + D$ features are input to the fully connected hidden layer 1 in Figure 2. At the fully connected hidden layer 1, the input shape matrix is of size $N*(K_U + K_I + K_C + D)$. The architecture of Layer 1 and Layer 2 is the same with an affine transformation followed by a non-linear activation layer namely a $Leaky\ ReLU$ and a $dropout$ layer. At each fully connected hidden layer H, W_H denotes the weight matrix, X_{H-1} denotes the input matrix to the hidden layer (which is the output of the previous hidden layer) and b_H is the bias of the hidden layer. In this case, the output of the linear layer is provided in (2) as the sum of the bias of the hidden layer and the matrix product of the weights of the hidden layer and the inputs to the hidden layer.

$$W_H . X_{H-1} + b_H \tag{2}$$



Journal Pre-proof

Figure 2. Proposed Deep Learning Solution Architecture

For example: for the first hidden layer, the input matrix shape for a single instance is $1*\left(K_U + K_I + K_C + D\right)$ and the output matrix shape is $1*h_1$, where h_1 is the number of hidden nodes in layer 1. The weight matrix at hidden layer 1 has a shape of $\left(K_U + K_I + K_C + D\right)*h_1$. Generalizing, the shape of the matrix at a hidden layer h is $h_{i-1}*h_i$, for the hidden layer i. The linear function above is followed by a non-linear activation. The non-linear activation employed is a $LeakyReLU$ function denoted by (3)

x= $LeakyReLU$ (x) (3)

$LeakyReLU$ is a function with the following form as shown in (4).

$$LeakyReLU(x) = max(x,0.01) \quad (4)$$

The $ReLU$ activation function has the advantage that its derivative is easy to compute and is currently the most popular non-linear activation function used in Deep Learning (Lecun et al., 2015). $LeakyReLU$ shares the ease of computation of $ReLU$ with the advantage that it does not suffer from the vanishing gradients problem. Vanishing gradients refers to the case where the gradients become zero or very close to zero preventing the network from learning. $LeakyReLU$ provides a lower bound on the value of the computed gradients. The non-linear $LeakyReLU$ activation is followed by a dropout function that takes as input a dropout probability p and randomly turns off p activations at its output. In the proposed solution, there are two fully connected hidden layers, each of which applies an affine linear transformation followed by a non-linear $LeakyReLU$ activation followed by a dropout. The third hidden layer just applies a $LeakyReLU$ function using the equations in (3) and (4) above. The output of the last layer is connected to a sigmoid function that forces the output to be between 0 and 1 using the sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad (5)$$

The final equation at the end of the sigmoid transformation is shown in (6). It shows how the proposed solution blends multiple linear and non-linear functions.

$$
\begin{aligned}
X_1 &= dropout_1(LeakyReLU(W_1 * X_0 + b_1)) \\
X_2 &= dropout_2(LeakyReLU(W_2 * X_1 + b_2)) \\
X_3 &= dropout_3(LeakyReLU(W_3 * X_2 + b_3)) \\
\sigma_{output} &= \sigma(X_3)
\end{aligned}
\tag{6}
$$

The above is a short form of the below

$$
\begin{aligned}
\sigma_{output} &= \sigma(dropout_3(LeakyReLU(W_3 * dropout_2(LeakyReLU * \backslash \\
&(W_2 * (dropout_1(LeakyReLU(W_1 * X_0 + b_1))) + b_2)) + b_3))
\end{aligned}
\tag{7}
$$

A linear transformation is applied on top of (6) to ensure that the final rating remains between the minimum observed rating denoted as $\min_{rating}$ and $\max_{rating}$ respectively as shown in (8). This ensures that the predicted rating is within the bounds of the observed ratings.

$$
output = \min_{rating} + \sigma_{output} * (\max_{rating} - \min_{rating}) \tag{8}
$$

The proposed deep learning architecture solves the optimization problem of minimizing the error at the output between the predicted and the actual values and the same is achieved by using stochastic gradient descent algorithm. Table 3 below lists the different hyper-parameters, how they influence the quality of the solution and the range of hyper-parameters employed by the solution.

Table 3. Hyper-parameters in the Proposed Solution

Hyper-parameter	**Influence on the solution**	**Recommended Setting/Approach Employed**
Learning Rate	Slower learning rate = Longer time to converge Higher learning rate = Miss the minima	Learning rate cyclically decayed between values 1e-2 and 1e-5. The exact values are found by the performance on the holdout set
Weight Decay	Higher weight decay = higher regularization and lower overfitting but lower accuracy. This is a form of L2-regularization	Usually 1e-4 but optimized for each dataset like any other hyper-parameter
Dropout	Higher dropout=higher regularization and lower overfitting but lower accuracy	Usually 0.25 but optimized like any other hyper-parameter
Batch Size	Smaller batch size=faster convergence but longer training time per epoch	64

3.4 Training the Deep Neural Network

There are two major challenges that the proposed solution needs to overcome, namely finding optimal learning rates for such a deep network and controlling overfitting as such a complex network

Journal Pre-proof

could easily fail to generalize on the validation dataset. A combination of regularization, early stopping and dropout techniques is used to control overfitting. Each layer has a dropout, starting from the categorical embeddings to both the fully connected hidden layers that follow the input layer. Additional regularization is achieved by using the weight decay parameter in the Adam optimizer (Kingma & Ba, 2015) that is used for stochastic gradient descent. Early stopping is employed by observing the validation error after each epoch and stopping the process of training the model when there are two consecutive epochs with no improvement in the solution. This solution incorporates two of the most recent developments in tuning a deep neural network namely the learning rate finder and stochastic gradient descent with restarts (SGDR). Both these methods are explained in this sub-section. Learning rate is easily the most important parameter to tune in a deep learning network and determines the impact that the gradient of loss has on the updating the weights. If the learning rate is too high, the solution could miss the optimal point and if it is too low, it may never converge in a timely manner or may get stuck in a local minima (Sutskever, Martens, Dahl, & Hinton, 2013). In normal stochastic gradient descent, a single learning rate is set at the beginning of the training process and a single learning rate is used to update all the parameters and is gradually decayed in each subsequent epoch (Bottou, 2010). The prominent optimizers of today like RMSProp (Dauphin, de Vries, & Bengio, 2015) and Adam (Kingma & Ba, 2015) adapt learning rates for each of the parameters being trained. To find the optimal learning rates, the proposed solution starts with a very low learning rate and increases the learning rate exponentially in each subsequent batch. The resulting losses are then plotted against the learning rates and the solution picks the optimal learning rate as the point where the loss is still decreasing as shown in Figure 3. While we decrease learning rate in every cycle across epochs, with the idea that the algorithm needs to take smaller steps as it gets closer to the minima but can take large steps initially, we also increase the regularization amount or the weight decay in every iteration to control overfitting.

Journal Pre-proof

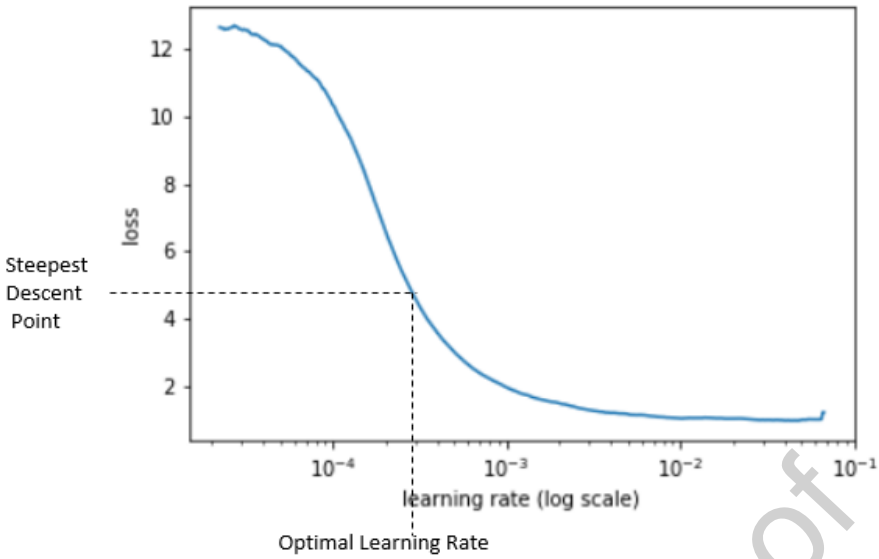


Figure 3. Finding the optimal learning rate by computing loss against exponentially increasing learning rate by batch

The proposed solution employs stochastic gradient descent with restarts (Loshchilov & Hutter, 2017), cyclical learning rates (Smith, 2017b) and weight decay concepts to further optimize the solution. An epoch is defined as a complete batch-wise run on the dataset and a cycle is defined as a group of epochs. In this solution, the learning rates are cycled from a maximum value to a minimum value in each cycle. The proposed method increases the cycle length by a multiplicative factor each time. An illustration of the pseudocode used to generate an exponentially decreasing learning rate is illustrated in Table 4. A similar approach in a direction opposite to the algorithm in Table 4 is employed for the weight decay, which is a regularization parameter. Weight decay ensures the weights get smaller and this is a mechanism to prevent overfitting. The pseudocode refers to the exponential decay of learning rate. lr_mult in the equation determines the amount by which the learning rate drops in each iteration. The pseudocode ensures that the decay rate is exponential between the values of the initial and final learning rates.

Table 4. Pseudocode for Cyclical Learning Rate Implementation

```
Input:
   learning rate 'lr',
   number of iterations 'iter'
Output:
   An array of learning rates to be applied per batch 'lrs'
begin
   base_lr = lr*0.1
   max_lr = lr * 10
```

```
    ratio = max_lr/base_lr
    lr_mult =ratio ** (1/iterations)
    lrs=[]
    for t in range (iter):
        lr*=lr_mult
        lrs.append(lr)
end
```

4 Experimental Setup and Test Bed

In this section, we outline the computing environment which is the hardware, software packages, datasets used and the validation dataset creation methodology. A brief exploratory analysis of the characteristics of the different datasets considered for experimentation is included in this section. The evaluation criteria and implementation of the competing techniques are also outlined in this section.

4.1 Computing Environment

All the experiments were performed on a desktop that has 64 GB RAM, 8th generation intel i7-8700 CPU, 2 Nvidia RTX 2080 GPUs (1 MSI GPU and 1 EVGA GPU) running Ubuntu 18.04. Python 3.7.3 was the programming language for the experimentation and the programming environment was a Jupyter Notebook. PyTorch 1.1.0 library was used for training the deep learning models. The python libraries like scikit-learn 0.20.3, numpy 1.16.2 and pandas 0.24.2 were extensively leveraged. We have made our implementation available for reproducing the results[2].

4.2 Datasets Used

For the purpose of experimentation, several datasets were considered and are described in this section. For book recommendations, the Book-Crossing (Ziegler et al., 2005) dataset was utilized. This dataset consists of ratings provided to books by users. Specifically, it contains 278,858 users and 1,149,780 ratings about 271,379 books. The rating scale is from 0 to 10 with the higher score indicating higher preferences. The dataset has side information about users (like location, age etc.) and items (like book title, author, year of publication, publisher etc.). The MovieLens 100K (ML100K) and the MovieLens 1M (ML1M) datasets are widely used in education, research and industry to benchmark different datasets. Their huge popularity is seen through 7500+ references to MovieLens in Google Scholar, 140000 downloads in 2014, 2750+ results in Google Books in 2014 and their presence in several MOOC courses across the world (Harper and Konstan, 2015). They consist of $<user, item, rating, timestamp>$ tuples of ratings of users to movies at a particular timestamp on a 0-5 scale in multiples of 0.5. These ratings come from the MovieLens website, which uses the ratings of users to provide personalized movie recommendations. The ML100K and the ML1M datasets are a

[2] https://github.com/efpm04013/dnnrec_eswa

widespread benchmark and are highly sparse datasets. The ML100K dataset consist of 100K ratings applied to 9742 movies by 610 users. The dataset contains side information about items (movie genres, title, user tags) but no side information about users. The ML1M dataset has 1 million ratings from 6000 users on 4000 movies. The ML1M dataset contains side information about users (gender, age, occupation, zip-code) and about items (Movie titles, genres etc.). The FilmTrust dataset (Guo et al., 2013) is also a movie recommender dataset with 35,497 ratings and comes from a popular movie sharing and ratings website. The scale of the ratings is 1-5. No preprocessing was performed on these datasets, except the Book-Crossing dataset where the rating value of 0 translated to an implicit rating and the records with such a rating were removed. Table 5 presents the summary of the characteristics of these datasets like the number of users, the number of items, the number of ratings, the scale of the ratings and the sparsity of these datasets. These datasets come from varying domains like book recommendations, movie recommendations and trust information. These datasets have different complexity levels and provide a test bed to evaluate the generic applicability of the proposed solution. The side information available in these datasets has already been provided in Table 2 in Section 3.

Table 5. Datasets used for Experimentation

Dataset	**Users**	**Items**	**Ratings**	**Sparsity**	**Scale**
Book-Crossing	278,858	271,379	1,149,780	99.99%	[0-10]
ML100K	610	9,742	100,836	93.7%	[1-5]
ML1M	6,000	4,000	1,000,000	95.8%	[1-5]
FilmTrust	1,508	2,071	35,497	98.86%	[1-5]

4.3 Creation of Validation Dataset

4.3.1 Classical Holdout and Cross-Validation approaches

The goal in a recommender system is to provide movie recommendations for the users based on the available history of prior ratings. The normal holdout set creation technique to accomplish this involves keeping a few ratings for each user as training and the rest as validation. This paper refers to the normal holdout dataset creation technique as the classical holdout approach. The normal cross-validation approach involves dividing the dataset into a set of folds, each with an equal number of ratings per user in each fold and using all but one fold to generate predictions on the remaining fold and repeating the process as many times as the number of folds. We refer to this approach as the classical cross-validation approach. In the classical holdout approach, there is no user cold start as the same users are present in both training and validation datasets. For the same reason, in the classical cross-validation approach, the

user cold start problem does not exist. In both these approaches, the item cold start problem could exist. Hence there is a need to simulate only the user cold start problem and not the item cold start problem.

4.3.2 Simulating the cold start problem

We note that in both the classical holdout and classical cross-validation approaches highlighted in section 4.3.1, the item cold start problem can exist. Hence we only need to simulate the user cold start problem. To simulate the cold start problem, there are several approaches to select a subset of users as training and the remaining as validation. We leverage one such approach where the users are randomly sampled into training and validation buckets (Basiri, Shakery, Moshiri, & Hayat, 2010; Son, 2016). To simulate the cold start problem, there are approaches to select a subset of users as training and the remaining as validation. (Basiri et al., 2010; Son, 2016). We apply this approach to a small portion of our dataset, namely the users acquired in the last year of the dataset, in cases where our dataset has timestamp information like the MovieLens datasets. In cases like Book-Crossing dataset and FilmTrust dataset, we use a random 10% of the users to simulate the cold start case similar to the MovieLens case. The cross-validation approach discussed above is represented diagrammatically in Figure 4 for the datasets with timestamp information. We see that there are two parts to the approach. For the users acquired in the last year, a random 10% of them are bucketed into validation and the remaining into training, thereby simulating the user cold start problem. For the users in the prior years, the classical holdout approach outlined in Section 4.3.1 is followed. In this way, user cold start is simulated into the datasets with timestamp information. This approach in Figure 4 was followed for the MovieLens 100K and MovieLens 1M datasets. For the datasets without timestamp information like Book-Crossing dataset and FilmTrust, similar approach is followed with the ‘last year’ replaced with a ‘random 10%’.

Journal Pre-proof

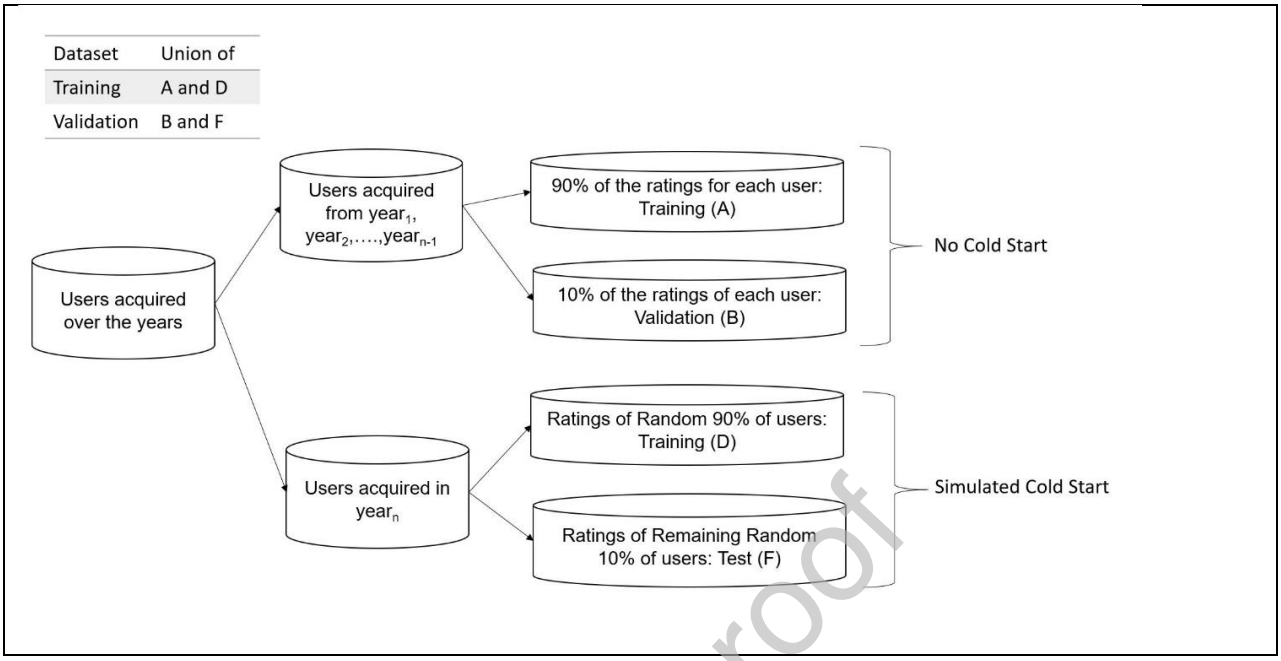


Figure 4. Cold Start Simulation for Datasets with Timestamp information

Figure 5 illustrates the cold start simulation for datasets without timestamp information.

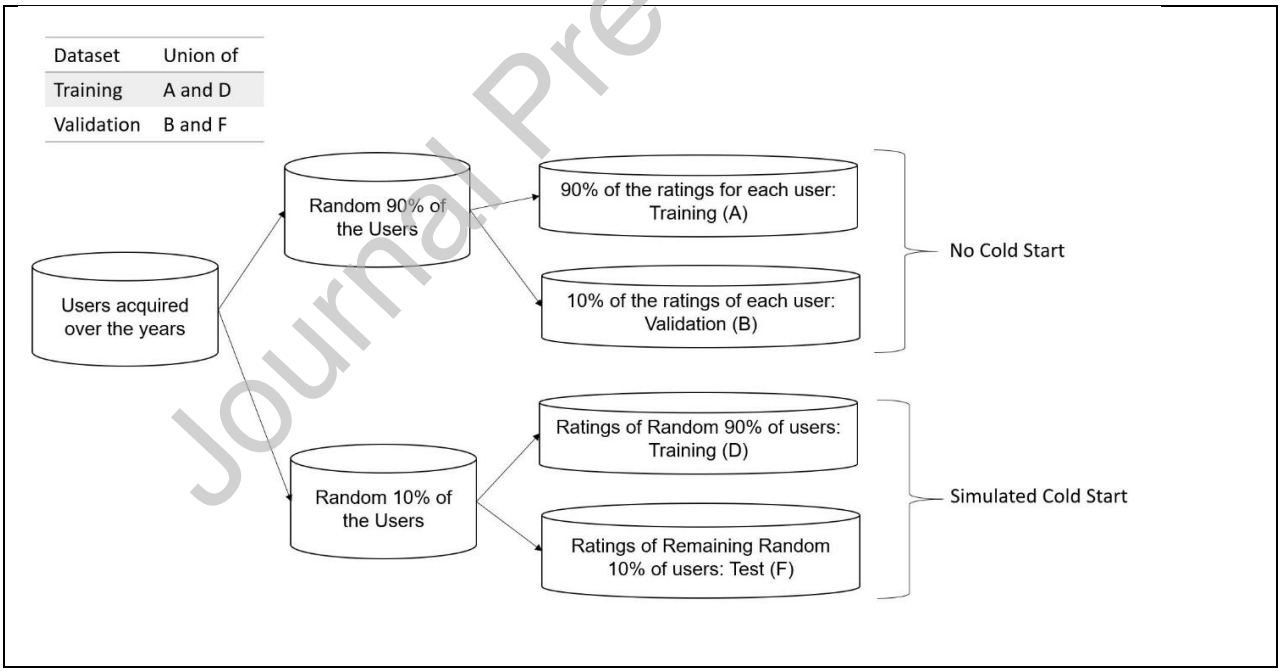


Figure 5. Cold Start Simulation for Datasets with Timestamp information

The percentage of observations in the validation dataset of the MovieLens 100K, FilmTrust, Book-Crossing, MovieLens 1M datasets across the non-cold start and cold start cases is shown in Table 6. As

we see the cold start distribution was highest in the Book-Crossing dataset. This is one of the reasons why the benchmarks reported for Book-Crossing dataset in literature are lower than the other datasets as the cold start problem presents difficulties in predictive accuracy.

Table 6. Distribution of Observations across the cold start cases

Dataset	No Cold Start	Cold Start
MovieLens 100K	91.8%	8.2%
FilmTrust	89.9%	10.1%
Book-Crossing	56.3%	43.7%
MovieLens 1M	91.13%	8.97%

4.4 Evaluation Criteria

The solutions are evaluated on both optimizing and satisficing criteria. For optimizing criteria, four evaluation metrics Root Mean Squared Error (RMSE), Mean Squared Error (MSE), Mean Absolute Error (MAE) and R-squared values are used to measure the predictive accuracy of the recommender system. For satisficing criteria, the mean and standard deviation of the running times of the algorithms across seven runs are computed. These measures are reported separately by overall and the cold start cases. The mathematical definitions of the evaluation metrics namely MSE, RMSE and MAE are in the equations (9) to (11) below. For these equations, T_{ij} is a matrix with a cell value t_{ij} = 1 only if the user i rated item j and 0 otherwise. r_{ij} denotes the rating provided by user i to item j.

$$RMSE = \sqrt{\sum_{i,j}^{m,n} T_{ij} (r_{ij}^{actual} - r_{ij}^{predicted})^2} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(9)$$

$$MSE = \sum_{i,j}^{m,n} T_{ij} (r_{ij}^{actual} - r_{ij}^{predicted})^2 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(10)$$

$$MAE = \sum_{i,j}^{m,n} T_{ij} \mid r_{ij}^{actual} - r_{ij}^{predicted}) \mid \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(11)$$

5 Results and Discussion

The experimental results on the different datasets are detailed in this section. Table 7 shows the RMSE, MSE, MAE and R-squared values for the FilmTrust, Movie Lens 100K, Book-Crossing and MovieLens 1M datasets. As we see DNNRec outperforms the other techniques on all the four datasets. For the ML100K dataset, the MSE, RMSE, MAE and R-squared of DNNRec are 1.2%, 6.27%, 0.7% and

2.5% better than the next best technique respectively. For FilmTrust dataset, the values are 2% 1%, 0.5% and 7.8% higher for DNNRec. For Book-Crossing, the values are 4.8%, 2.4%, 2.2% and 4.7% higher for DNNRec. For ML1M, the outperformance is 1.65%, 0.83%, 0.47% and 1.24% respectively. As we see, DNNRec outperforms other techniques on all the four metrics on all the four datasets.

Table 7. MSE, RMSE, MAE and R-squared on the ML100K, Book-Crossing, ML1M and FilmTrust datasets

Dataset	Method	MSE (lower is better)	RMSE (lower is better)	MAE (lower is better)	R-squared (higher is better)
MovieLens 100K	Global Average	1.129	1.062	0.842	0.000
	User Average	0.929	0.964	0.752	0.177
	Item Average	1.010	1.005	0.782	0.105
	User KNN – Cosine	0.813	0.813	0.813	0.279
	Item KNN - Cosine	0.854	0.862	0.661	0.241
	User KNN – Pearson	0.814	0.902	0.693	0.279
	Item KNN - Pearson	0.796	0.892	0.686	0.295
	SVD	0.879	0.937	0.718	0.221
	Biased MF	0.768	0.877	0.675	0.319
	SVD++	0.756	0.869	0.671	0.330
	Non-deep NN	0.779	0.882	0.691	0.310
	DNNRec	**0.747**	**0.864**	**0.666**	**0.338**
FilmTrust	Global Average	0.837	0.915	0.716	-0.001
	User Average	0.702	0.838	0.654	0.161
	Item Average	0.837	0.915	0.723	-0.001
	User KNN – Cosine	0.704	0.839	0.655	0.159
	Item KNN - Cosine	0.662	0.814	0.629	0.208
	User KNN – Pearson	0.704	0.839	0.655	0.159
	Item KNN - Pearson	0.662	0.814	0.629	0.208
	SVD	0.718	0.847	0.652	0.142
	Biased MF	0.672	0.820	0.639	0.197
	SVD++	0.681	0.825	0.659	0.185
	Non-deep NN	0.680	0.824	0.659	0.187
	DNNRec	**0.649**	**0.805**	**0.626**	**0.225**
Book-Crossing	Global Average	3.454	1.859	1.509	0.000
	User Average	2.938	1.714	1.309	0.149
	Item Average	3.812	1.952	1.537	-0.104

Journal Pre-proof

	User KNN – Cosine	3.107	1.763	1.361	0.100
	User KNN – Pearson	3.263	1.851	1.429	0.105
	SVD	3.734	1.932	1.537	-0.081
	Biased MF	3.166	1.779	1.353	0.083
	SVD++	2.898	1.702	1.330	0.161
	Non-deep NN	3.411	1.847	1.472	0.012
	DNNRec	**2.759**	**1.661**	**1.280**	**0.169**
ML1M	Global Average	1.332	1.154	0.954	-0.019
	User Average	1.174	1.083	0.863	0.102
	Item Average	0.985	0.992	0.790	0.247
	User KNN – Cosine	0.851	0.923	0.724	0.349
	Item KNN - Cosine	0.827	0.909	0.713	0.367
	User KNN – Pearson	0.838	0.915	0.719	0.359
	Item KNN - Pearson	0.803	0.896	0.700	0.389
	SVD	0.806	0.898	0.707	0.384
	Biased MF	0.782	0.884	0.696	0.402
	SVD++	0.779	0.882	0.691	0.404
	Non-deep NN	0.821	0.928	0.730	0.422
	DNNRec	**0.766**	**0.875**	**0.688**	**0.417**

Table 8 shows the RMSE, MSE, MAE and R-squared values for the Cold start case to compare and contrast the performance. As we see DNNRec outperforms the other techniques on all the four datasets in the cold start case. For the ML100K dataset, the MSE, RMSE, MAE and R-squared of DNNRec are 6.12%, 3.11%, 3.54% and 12.82% better than the next best technique respectively. For FilmTrust dataset, the values are 1.3%, 0.66%, 2.85% and 23% higher for DNNRec. For Book-Crossing, the values are 8.4%, 4.3%, 3.6% and 2% higher for DNNRec. For ML1M, the outperformance is 18.4%, 9.7%, 11.1% and 64% respectively. As we see, DNNRec outperforms other techniques on all the four metrics on all the four datasets in the cold start case also

Table 8. MSE, RMSE, MAE and R-squared on the ML100K, Book-Crossing, ML1M and FilmTrust datasets for the cold start case

Dataset	Method	MSE (lower is better)	RMSE (lower is better)	MAE (lower is better)	R-squared (higher is better)
MovieLens 100K	Global Average	1.411	1.188	0.966	-0.026
	User Average	0.939	0.969	0.773	0.317
	Item Average	1.360	1.166	0.940	0.011
	User KNN – Cosine	1.094	1.046	0.909	-0.191

Journal Pre-proof

	Item KNN - Cosine	0.983	0.991	0.801	0.285
	SVD	1.411	1.188	0.966	-0.026
	Biased MF	1.008	1.004	0.808	0.267
	SVD++	0.931	0.965	0.778	0.323
	Non-Deep NN	0.970	0.985	0.805	0.294
	DNNRec	**0.874**	**0.935**	**0.746**	**0.364**
FilmTrust	Global Average	1.024	1.012	0.807	0.000
	User Average	1.036	1.018	0.807	-0.012
	Item Average	0.973	0.986	0.786	0.049
	User KNN – Cosine	0.968	0.984	0.791	0.054
	Item KNN - Cosine	0.983	0.991	0.777	-0.054
	SVD	1.024	1.012	0.807	0.000
	Biased MF	0.971	0.986	0.786	0.051
	SVD++	0.974	0.987	0.794	0.048
	Non-Deep NN	1.041	1.020	0.865	-0.018
	DNNRec	**0.956**	**0.978**	**0.799**	**0.066**
Book-Crossing	Global Average	3.623	1.903	1.547	-0.004
	User Average	3.075	1.753	1.348	0.148
	Item Average	3.769	1.941	1.557	-0.044
	User KNN – Cosine	3.025	1.739	1.357	0.162
	Item KNN – Cosine	3.232	1.798	1.418	0.055
	SVD	3.623	1.903	1.547	-0.004
	Biased MF	3.013	1.736	1.328	0.165
	SVD++	3.103	1.762	1.380	0.140
	Non-Deep NN	4.116	2.029	1.637	-0.140
	DNNRec	**2.759**	**1.661**	**1.280**	**0.169**
ML1M	Global Average	1.198	1.094	0.918	-0.002
	User Average	1.197	1.094	0.917	-0.001
	Item Average	0.969	0.985	0.790	0.189
	User KNN – Cosine	0.968	0.984	0.787	0.190
	Item KNN - Cosine	0.968	0.984	0.787	0.190
	SVD	1.198	1.094	0.918	-0.002
	Biased MF	0.976	0.988	0.788	0.184
	SVD++	0.966	0.983	0.790	0.192
	Non-Deep NN	0.943	0.964	0.776	0.242
	DNNRec	**0.788**	**0.888**	**0.700**	**0.397**

Journal Pre-proof

Table 9 shows the running times as $mean \pm stdev$ across 7 runs for the different datasets. The proposed Deep Learning Solution, though performing better, takes a longer training time than all methods except the Nearest Neighbor methods. The GPU running time of the deep learning solution is comparable to the CPU running time of the matrix factorization methods. Since GPU and CPU are not comparable, given the rapidly decreasing hardware costs and increasing availability of GPUs, it is observed that that the 'running time' for the deep learning algorithm can be within acceptable limits. Given the prevalence of GPUs and the ability to run complex matrix computations in parallel, the training time taken by the deep learning methods seems acceptable when compared to matrix factorization. From an industry and real-world applicability, the prediction/recommendation times of DNNRec are important. The same is provided in Table 10. The prediction/recommendation task is an application of a sequence of affine followed by non-linear transformations on the data. Another observation is that the training time is heavily influenced by the size of the training dataset as we see in Table 9. The datasets with more instances like Book-Crossing, ML1M take more time to train than smaller datasets like ML100K and FilmTrust. The architecture we have used for each of these datasets is common and to retrain new models, the running times would be influenced by the size of the dataset. This model is similar to other machine learning models where retraining is required when the input characteristics change or substantially new data is available. We recommend that the output be periodically tested against new data to see if the holdout predictive accuracy has degraded. At that point in time, retraining the model would become necessary.

Table 9. Training Time Comparison across different datasets

	ML100K	ML100K	FilmTrust	FilmTrust	Book-Crossing	Book-Crossing	ML1M	ML1M
Method	Mean time of run across 7 runs (ms)	Standard deviation of time across 7 runs (ms)	Mean time of run across 7 runs (ms)	Standard deviation of time across 7 runs (ms)	Mean time of run across 7 runs (ms)	Standard deviation of time across 7 runs (ms)	Mean time of run across 7 runs (ms)	Standard deviation of time across 7 runs (ms)
Global Average	3.5	0.1	1.2	0.0	7.7	0.2	25.3	0.6
User Average	6.6	0.6	2.3	0.2	19.1	1.6	63.3	5.2
Item Average	7.3	0.7	2.6	0.2	21.1	1.9	52.2	9.3
User KNN	41500.0	637.0	14609.1	224.2	89717.5	459.0	444868.6	2276.2
Item KNN	520000.0	10100.0	183054.1	3555.5	749447.7	29113.2	1858085.7	72179.5
SVD++	1340.0	26.1	471.7	9.2	5793.8	37.6	19152.6	93.3
Biased MF	1930.0	19.1	679.4	6.7	8344.8	82.6	20689.1	409.5
SVD	1530.0	491.0	538.6	172.8	6615.3	2123.0	21868.2	5263.4
Non-Deep NN	1350.0	991.0	475.2	348.9	3891.4	2856.5	9647.8	9442.9

Journal Pre-proof

DNNRec (on GPU)	1980.0	118.4	697.0	41.7	4280.5	170.6	10612.5	846.1

a

Table 10. Prediction/Recommendation Times of DNNRec

		ML100K	**FilmTrust**	**FilmTrust**	**Book-Crossing**	**Book-Crossing**	**ML1M**	**ML1M**
Method	Mean time of run across 7 runs (ms)	Standard deviation of time across 7 runs (ms)	Mean time of run across 7 runs (ms)	Standard deviation of time across 7 runs (ms)	Mean time of run across 7 runs (ms)	Standard deviation of time across 7 runs (ms)	Mean time of run across 7 runs (ms)	Standard deviation of time across 7 runs (ms)
DNNRec (on GPU)	560	28	197	10	1211	81	5003	108

As we see, the proposed deep learning solution DNNRec outperforms the other techniques on both overall and cold start cases on the full spectrum of evaluation metrics for the ratings prediction case of recommender systems namely MSE, RMSE, MAE and R-squared. In this context, the weightage of predictive accuracy is higher than training time as the problem of ratings recommendations is not a streaming problem where the goal is to get a good solution as fast as possible. By running on a GPU, using batch stochastic gradient descent, it is possible to run massive matrix computations in parallel. As a result, the speedup through a GPU lends training times that are comparable to matrix factorization methods that are run on a CPU. The proposed deep learning solution out-performs neighborhood based collaborative filtering methods (user-KNN and item-KNN), heuristic methods (Global Average, Global User Average, Global Item Average) and matrix factorization based collaborative filtering methods (BiasedMF, Collaborative Filtering, SVD, SVD++). We have employed a non-deep NN for comparison as well with just a single hidden layer in Tables 8 and 9 and we see that DNNRec out-performs the same. Hence we can say that the depth of the network and the training interventions applied via the cyclical learning rates have the desired effect. A very deep network is able to learn very complex features that provide the high predictive accuracy. The proposed novel deep learning architecture can leverage both the content based-features (off the side information from the users and items) and exploit the relationship between users and items that is encoded in the ratings matrix.

6 Conclusion and Future Scope of Work

In this work, a hybrid recommender system using deep learning (DL) approach was proposed. The proposed solution leverages a complex architecture with a very deep neural network, uses embeddings to learn non-linear latent factors for users and items, combines the deep learning features with side information available about users and items to create a hybrid system, tunes the learning rates and weight

Journal Pre-proof

decay (regularization for comparable running times and prevents overfitting through a combination of dropout, regularization and early stopping. This paper provides a theoretical foundation for the proposed hybrid deep learning solution. The implementation results are compared against a plethora of techniques across heuristics, neighbor based, machine learning based and matrix factorization methods. All well-known recommender system solutions are compared on the benchmark MovieLens datasets on both predictive accuracy and running times. Further, the results for predictive accuracy are compared in both overall and cold start cases. The contributions in the paper are five-fold. First, the proposed novel solution beats the existing benchmarks of all existing methods across matrix factorization, heuristics, nearest neighbor and machine learning methods on both RMSE and R-squared. As a result, the solution presents a new state-of-the-art for recommender systems. Second, performance in the cold start case is improved by incorporating side information about users and items. The cold start case is one of the gaps in collaborative filtering systems and our solution addresses the same. Third, the embeddings in DNNRec allow the learning of non-linear latent features for users and items. This addresses the gap in CF methods that allow the learning of only linear latent user and item features. Fourth, the solution ensures reasonable running times on GPU by tuning learning rates resulting in faster and better convergence. Fifth, the proposed novel solution combines the state-of-the-art developments in deep learning like embeddings, Stochastic Gradient Descent with Restarts, cyclical learning rates, very deep network, batch normalization, dropouts and early stopping to improve predictive accuracy.

One limitation of the study is that it addresses the ratings prediction case of the recommender systems problem and not the ranking case or the streaming case, which could be opportunities for future research. This paper in-order to compare the performance of the different algorithms in the cold start cases simulates the cold start problem in the datasets. This means that the reported benchmarks of different algorithms are no longer comparable and these algorithms need to be run again on the new dataset. As a result, we ran all the matrix factorization, collaborative filtering, global average and nearest neighbors methods on the newly created datasets. In cases, where public implementations of benchmark techniques were unavailable, they could not be compared. For e.g. we would have liked to compare some of the autoencoder approaches listed in Section 2 if the exact public implementations for these methods were available. To ensure reproducible research on top of our work, we have made our code available on GitHub[3]. A last limitation of our study is the ability to provide a clear semantic explanation of differences in predicted ratings. White-boxing neural network predictions is tough and explaining the 'why' of predictions of deep neural networks is tougher. This could also be an area of future research.

[3] https://github.com/efpm04013/dnnrec_eswa

Journal Pre-proof

There are several opportunities for further research. In addition to the research areas arising from the limitations of the current study, we present allied and additional areas of research. The solution could be extended to the ranking case of recommender systems in addition to the prediction case. Another opportunity is to understand the value in bringing in additional side information about the items and users from the Internet. E.g. While the ML100K dataset has information about the title, genre and tag information of the movies, the trailer information for these movies could be downloaded from YouTube and run through a convolutional neural network (CNN) to derive additional features. Similarly, textual information like reviews of Books in the Book-Crossing dataset. Such additional information could alleviate the cold start problem further. Such a study could be a pointer to industry and academia to the actual worth of adding side information in this manner. We have considered different matrix factorization, nearest neighbor and global averages methods in this section for comparison. A third future research opportunity could be to understand the incremental value in predictive accuracy that an ensemble of techniques could bring. Given the varied nature of the techniques applied namely heuristics, neighborhood methods, matrix factorization methods, machine learning methods and deep learning method, there could be benefits realized by creating an ensemble of these techniques. A fourth research area could be to improve the training and recommendation times of the algorithm by looking at different and more effective tuning of hyper-parameters like learning rate, momentum and weight decay to lead to faster convergence. Lastly, a framework based on deep neural network, a novel architecture that exploits the sequential nature of the recommendations using recurrent neural networks could be employed.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Journal Pre-proof

References

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*. https://doi.org/10.1109/TKDE.2005.99

Aggarwal, C. C. (2016). An Introduction to Recommender Systems. In *Recommender Systems*. https://doi.org/10.1007/978-3-319-29659-3_1

Araque, O., Corcuera-Platas, I., Sánchez-Rada, J. F., & Iglesias, C. A. (2017). Enhancing deep learning sentiment analysis with ensemble techniques in social applications. *Expert Systems with Applications*. https://doi.org/10.1016/j.eswa.2017.02.002

Balakrishnan, A. (2014). DeepPlaylist : Using Recurrent Neural Networks to Predict Song Similarity. *Stanfort University*, 1–7. Retrieved from https://cs224d.stanford.edu/reports/BalakrishnanDixit.pdf

Bansal, T., Belanger, D., & McCallum, A. (2016). Ask the GRU. In *Proceedings of the 10th ACM Conference on Recommender Systems - RecSys '16* (pp. 107–114). https://doi.org/10.1145/2959100.2959180

Basiri, J., Shakery, A., Moshiri, B., & Hayat, M. Z. (2010). Alleviating the cold-start problem of recommender systems using a new hybrid approach. *2010 5th International Symposium on Telecommunications*. https://doi.org/10.1109/ISTEL.2010.5734161

Bernardi, L., Kamps, J., Kiseleva, J., & Mueller, M. J. I. (2015). The continuous cold start problem in e-commerce recommender systems. In *CEUR Workshop Proceedings* (Vol. 1448, pp. 30–33).

Bottou, L. (2010). Large-Scale Machine Learning with Stochastic Gradient Descent. *Proceedings of COMPSTAT'2010*. https://doi.org/10.1007/978-3-7908-2604-3_16

Catherine, R., & Cohen, W. (2017). TransNets. In *Proceedings of the Eleventh ACM Conference on Recommender Systems - RecSys '17* (pp. 288–296). https://doi.org/10.1145/3109859.3109878

Covington, P., Adams, J., & Sargin, E. (2016). Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems - RecSys '16* (pp. 191–198). https://doi.org/10.1145/2959100.2959190

Dauphin, Y. N., de Vries, H., & Bengio, Y. (2015). RMSProp and Equilibrated adaptive learning rates for non-convex optimization. *NIPS '15*, 10. Retrieved from http://arxiv.org/abs/1502.04390

De Brébisson, A., Simon, É., Auvolat, A., Vincent, P., & Bengio, Y. (2015). Artificial neural networks applied to taxi destination prediction. In *CEUR Workshop Proceedings*. https://doi.org/10.1093/mnras/stv1765

Do, H. H., Prasad, P. W. C., Maag, A., & Alsadoon, A. (2019). Deep Learning for Aspect-Based Sentiment Analysis: A Comparative Review. *Expert Systems with Applications*. https://doi.org/10.1016/j.eswa.2018.10.003

Gantner, Z., & Rendle, S. (2011). MyMediaLite: A free recommender system library. *Proceedings of the Fifth ACM Conference on Recommender Systems*. https://doi.org/10.1145/2043932.2043989

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural Networks and the Bias/Variance Dilemma. *Neural Computation*. https://doi.org/10.1162/neco.1992.4.1.1

Goldberg, Y., Levy, O., Rong, X., Perozzi, B., Al-Rfou, R., Skiena, S., … Dean, J. (2014). word2vec Parameter Learning Explained Continuous Bag-of-Word Model. *Proceedings of the International Conference on Learning Representations (ICLR 2013)*. https://doi.org/10.1162/jmlr.2003.3.4-5.951

Gomez-Uribe, C. A., & Hunt, N. (2015). The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.*, *6*(4), 13:1--13:19. https://doi.org/10.1145/2843948

Guibing Guo, Jie Zhang, Z. S. and N. Y.-S. (2015). LibRec: A Java Library for Recommender Systems. *Proceedings of the 23rd Conference on User Modelling, Adaptation and Personalization (UMAP)*.

Harper, F. M., & Konstan, J. A. (2015). The MovieLens Datasets. *ACM Transactions on Interactive Intelligent Systems*. https://doi.org/10.1145/2827872

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). （NCF）Neural Collaborative Filtering. *WWW*, 173–182. https://doi.org/10.1145/3038912.3052569

Journal Pre-proof

Jannach, D., Lerche, L., Kamehkhosh, I., & Jugovac, M. (2015). What recommenders recommend: an analysis of recommendation biases and possible countermeasures. *User Modeling and User-Adapted Interaction*. https://doi.org/10.1007/s11257-015-9165-3

Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations 2015*, 1–15. https://doi.org/http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503

Koren, Y. (2008). Factorization meets the neighborhood. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*. https://doi.org/10.1145/1401890.1401944

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*. https://doi.org/10.1109/MC.2009.263

Lam, X., Vu, T., & Le, T. (2008). Addressing cold-start problem in recommendation systems. *Proceedings of the 2nd International*, 208–211. https://doi.org/10.1145/1352793.1352837

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*. https://doi.org/10.1038/nature14539

Lee, D., Huntsman, J. M., Fl, H., Huntsman, J. M., & Fl, H. (2014). Impact of Recommender Systems on Sales Volume and Diversity. *Icis*, 1–15.

Lops, P., de Gemmis, M., & Semeraro, G. (2011). Content-based Recommender Systems: State of the Art and Trends. In *Recommender Systems Handbook*. https://doi.org/10.1007/978-0-387-85820-3_3

Loshchilov, I., & Hutter, F. (2017). SGDR: Stochastic Gradient Descent with Restarts. *ICLR*. https://doi.org/10.1002/fut

Mikolov, T., Corrado, G., Chen, K., Dean, J., Lai, S., Liu, K., … Zweig, G. (2013). word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *Artificial Intelligence*. https://doi.org/10.1162/jmlr.2003.3.4-5.951

Netflix. (2009). Netflix Prize. *Proceedings of KDD Cup and Workshop*, *2007*(3/6), 3–6. https://doi.org/10.1145/1562764.1562769

Oord, A. van den, Dieleman, S., & Schrauwen, B. (2013). Deep content-based music recommendation. *Electronics and Information Systems Department (ELIS)*, 9. https://doi.org/10.1109/MMUL.2011.34.van

Salakhutdinov, R., Mnih, A., & Hinton, G. (2007). Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning - ICML '07* (pp. 791–798). https://doi.org/10.1145/1273496.1273596

Sarwar, B. M., Karypis, G., Konstan, J. a, & Riedl, J. T. (2000). Application of Dimensionality Reduction in Recommender System - A Case Study. *Architecture*. https://doi.org/10.1.1.38.744

Smith, L. N. (2017a). Cyclical learning rates for training neural networks. In *Proceedings - 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*. https://doi.org/10.1109/WACV.2017.58

Smith, L. N. (2017b). Cyclical Learning Rates for Training Neural Networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. https://doi.org/10.1109/WACV.2017.58

Son, L. H. (2016). Dealing with the new user cold-start problem in recommender systems: A comparative review. *Information Systems*. https://doi.org/10.1016/j.is.2014.10.001

Journal Pre-proof

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. https://doi.org/10.1109/ICASSP.2013.6639346

Wei, J., He, J., Chen, K., Zhou, Y., & Tang, Z. (2017). Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications*. https://doi.org/10.1016/j.eswa.2016.09.040

Zhang, D., Chen, X., Wang, D., & Shi, J. (2018). A survey on collaborative deep learning and privacy-preserving. In *Proceedings - 2018 IEEE 3rd International Conference on Data Science in Cyberspace, DSC 2018*. https://doi.org/10.1109/DSC.2018.00104

Journal Pre-proof

Appendix 1: Exploratory Analysis of ML100K dataset to illustrate the user cold start simulation for dataset with ratings

An exploratory analysis of the MovieLens 100K dataset is shared to explain the logic used in creating the validation dataset as this study has simulated the cold start problem. Figure 6 shows the distribution of ratings by year. For all years until the last year, the first few ratings for each user are used to generate user and movie features, the latter major part form the training set and the remaining form part of the validation set. To simulate cold start, the users in the last year are randomly sampled into training and validation datasets. This approach is explained in detail in Section 4.3.1

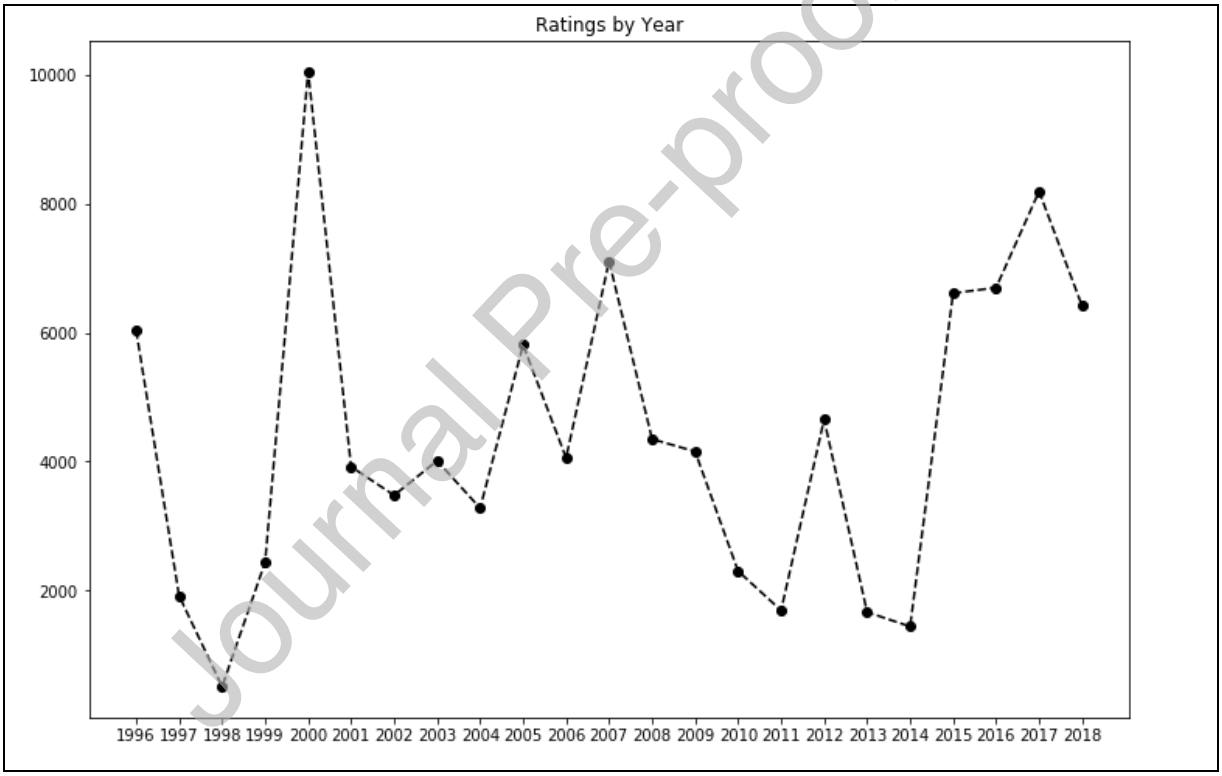


Figure 6. Ratings by Year for MovieLens 20M dataset

Figure 7 shows the distribution of the new users in the system. The users acquired in the last year are used to simulate the cold start problem like we outline in Section 4.3.1.

Journal Pre-proof

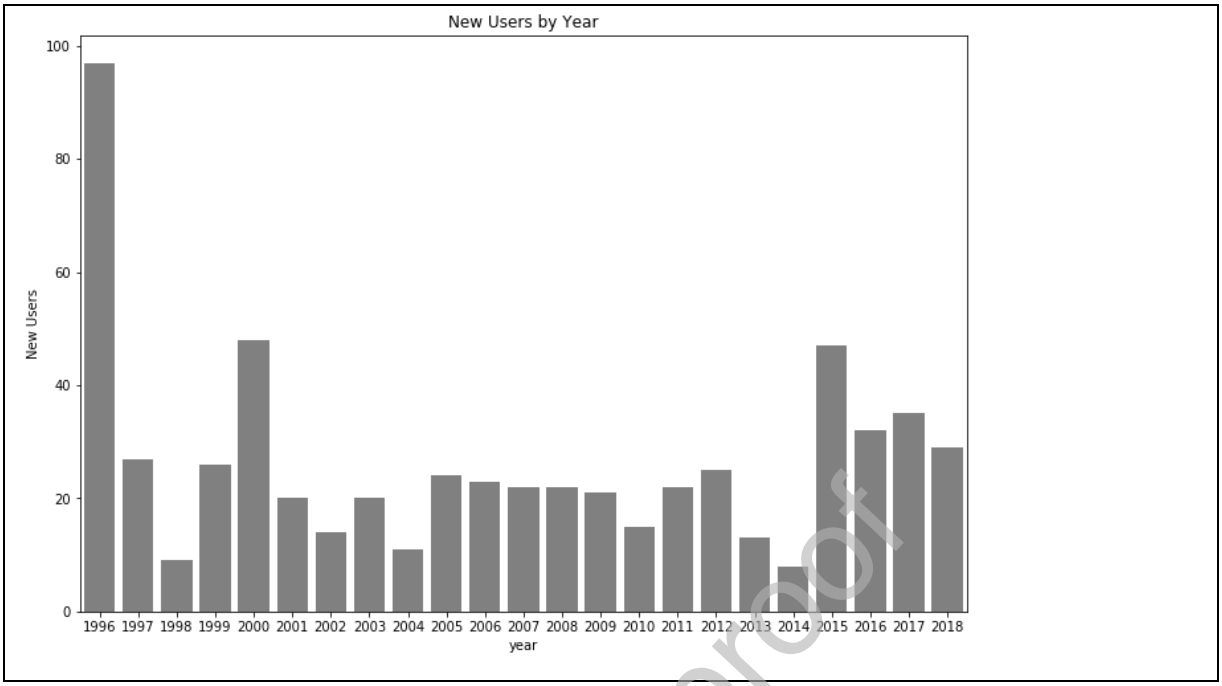


Figure 7. New Users by Year for MovieLens 20M dataset

Figure 8 shows the ratings by year and month

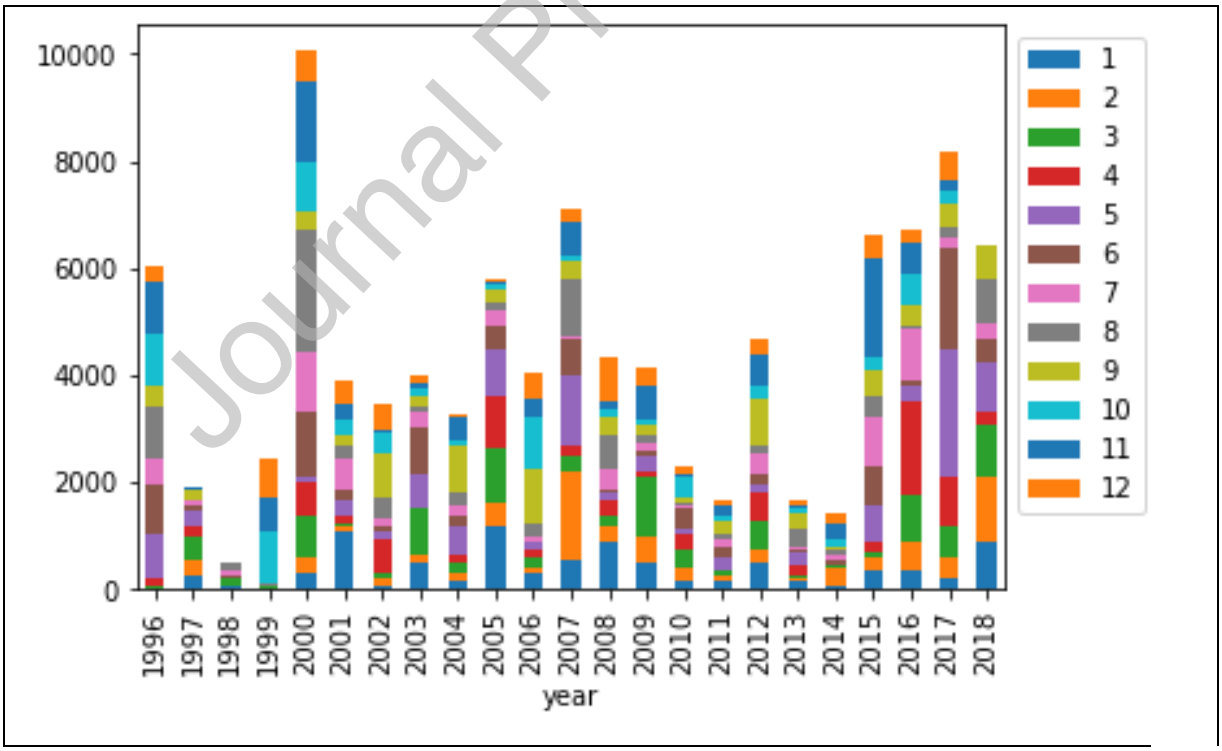


Figure 8. Ratings by Year and Month (Legend) for MovieLens 100K dataset

Journal Pre-proof