Dealing with cold-start problems in Recommender Systems: a Deep Neural Network-based method

Guannan Qu
STUDENT NUMBER: u174615

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN DATA SCIENCE & SOCIETY
DEPARTMENT OF COGNITIVE SCIENCE & ARTIFICIAL INTELLIGENCE
SCHOOL OF HUMANITIES AND DIGITAL SCIENCES
TILBURG UNIVERSITY

Thesis committee:
Supervisor: Dr. Gonzalo Nápoles
Second Reader: Dr. Görkem Saygili

Tilburg University
School of Humanities and Digital Sciences
Department of Cognitive Science & Artificial Intelligence
Tilburg, The Netherlands
June 2021

Dealing with cold-start problems in Recommender Systems: a Deep Neural Network-based method

Guannan Qu

Nowadays, Recommender Systems (RSs) have been ubiquitously used on the internet-based platforms to make recommendations for users. However, the mainstream recommendation algorithms such as Collaborating Filtering (CF) and Matrix Factorization (MF) are usually plagued by the new user and new item cold-start problems. The root of the problems is the lack of prior information of new users and new items that these mainstream algorithms rely on. Researchers have attempted to use various approaches to ameliorating the cold-start problems in RSs. However, most of them have only achieved to mitigate one of the cold-start cases, either the new user cold-start or the new item cold-start. Therefore, the main goal of this thesis is to fully utilize users' and items' observable side information to train a Deep Neural Network (DNN) based model, which can alleviate both new user and new item cold-start problems. The proposed model is trained and evaluated on MovieLens-1M dataset, and hence it is a movie recommender system. However, once auxiliary information of users and items is provided, the proposed model can be applied in various fields. Furthermore, to verify the effectiveness of the proposed model, several state-of-the-art recommendation algorithms will be adopted for comparison. The experimental results indicate that the proposed model can outperform all the compared algorithms in terms of the Mean Absolute Error (MAE) metric.

1. Introduction

As online information exponentially increases, information overload has become an unavoidable dilemma. A primary challenge for information producers is to efficiently provide users with the information that interests them from the overloaded information. To this end, Recommender Systems (RSs) have emerged to overcome this challenge by using data mining techniques. The essence of RSs is leveraging big data on consumers and products to generate customized dynamic recommendations. RSs are ubiquitously used in our daily life, such as in e-commerce, social networks, and entertainment (Da'u and Salim 2019). Thus, building an effective personalized recommender system is of importance for internet-based platforms to attract and retain customers and maximize the value of their products.

Content-based (CB) and collaborative filtering (CF) are two main classic approaches to building RSs (Wang, Jiang, and Lou 2020). CB recommender system makes recommendations by analyzing users' preferences for items' attributes based on users' historical behaviors. Similarly, CF recommender system also leverages users' browsing and evaluating history to generate recommendations, yet it tends to detect correlations or similarities between users and items from the user-item rating matrix. The system

then makes recommendations based on similar items or similar users. Both CB and CF methods are commonly used and have their own advantages. However, they are not without their downsides, such as data sparsity and cold-start problems (Barjasteh et al. 2016). Most users generate very limited feedback or ratings to the items produced, making the user-item matrix very sparse (see Table 1). Hence, CF method is incapable of extracting valid correlations or similarities between users and items to provide valuable recommendations. Another crucial issue for both CB and CF recommender systems is known as the cold-start problem. It includes the new user cold-start problem and the new item cold-start problem. The problems arise because there is no historical rating information of new users and new products. Therefore, both CB and CF recommender systems are not feasible in this situation.

Table 1
An example sparse user-item matrix presenting ratings (on a five-point scale) provided by K users to N items, where "?" denotes the missing ratings. It can hardly provide useful information like correlations or similarities between users and items.

Ratings	Item 1	Item 2	Item 3	Item 4	.....	Item N
User 1	2	?	?	3	...	5
User 2	?	3	?	3	...	?
User 3	4	4	?	?	...	2
User 4	?	1	3	?	...	4
.....	...	...	...	...	...	...
User K	2	?	5	2	...	?

Traditional CB and CF methods merely concentrate on historical information to generate recommendations (Son 2016), which causes the aforementioned problems. However, except for the data on users' browsing and evaluating history, side information of users and items can also be exploited in RSs to ameliorate these problems (Low, Yap, and Tee 2018). Such information includes but is not limited to users' demographic data and item attributes. These auxiliary data are not difficult to obtain (Gupta and Katarya 2020), as they are mostly provided when users register and items are launched online. However, these data can be extremely complex due to their high dimensionalities and multiple modalities (Gupta and Katarya 2020). To manage such massive and complex data, deep learning methods can be applied, as they have shown extraordinary strength in learning complex feature representations in various fields (Da'u and Salim 2019).

Thus, the main goal of this thesis is to utilize deep neural networks trained on side information of users and items to build a personalized recommender system that can mitigate both new user and new item cold-start problems. To achieve it, the main research question below is needed to be answered.

RQ: *How does the deep neural network leverage users' and items' side information to alleviate cold-start problems in Recommender Systems (RSs)?*

To tackle the main research question, I aim to train the deep neural networks on side information of users and items to build a movie recommender system by using

MovieLens-1M dataset (Harper and Konstan 2015). Once the side information of users (ID, age, gender, and occupation) and movies (ID, genre, title, and released year) are provided, they can be used as input features feeding in neural networks to predict the labels—ratings. Due to the full use of side information of users and movies, instead of only focusing on prior information, the proposed model can mitigate the cold-start problems that plague traditional CB and CF RSs.
Following the main research question come three sub research questions aiming to examine and improve the performance of the proposed model:

SQ1: *How hyperparameters, such as learning rate, dropout rate, and batch size, affect the accuracy of the proposed model?*

This sub question is proposed to find the optimal model for further model evaluations. To answer it, it needs to be examined whether different hyperparameter configurations can influence the model's performance and which configuration can yield the best result. Thus, hyperparameter optimization is needed to be implemented. However, due to the limited time and experimental hardware conditions, only three hyperparameters (i.e., learning rate, dropout rate, and batch size) are tuned for 30 trials based on Bayesian optimization method. The results show that different hyperparameter combinations indeed significantly impact the performance of the proposed model. Therefore, the model that generates the best result will eventually be selected to compare with several baselines and competitors.

SQ2: *Does the proposed model outperform other classic RSs models?*

The second sub question aims to further verify the effectiveness of the proposed model. To answer it, it needs to be examined whether and to what extent the proposed model can outperform the compared algorithms. In experiments, two baselines and three competitors are employed to compare with the proposed model in terms of the Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) metrics. The baselines are user's average rating and movie's average rating, while the competitors include user-based CF (UBCF), item-based CF (IBCF) and Singular Value Decomposition (SVD) algorithms. The comparisons are conducted on different sampling settings (i.e., 60%, 70%, 80%, and 90% training sizes). For each sampling setting, it is found that the proposed model is superior to the compared algorithms.

SQ3: *Instead of only utilizing observable side information, does adding extra latent features improve the proposed model's performance?*

Historical rating data are not fully used in the proposed model, while some useful latent features can be mined from them to further increase the accuracy of recommendations (Bi, Liu, and Fan 2020). Hence, the third sub question is proposed to examine whether the use of historical rating data can contribute to improving the model's performance. From another point of view, $SQ3$ also tries to investigate whether purely using side information as inputs while ignoring historical data in the proposed model will compromise the model's performance. Thus, except for observable side information, three extra features extracted from historical rating data are also fed into the proposed model as inputs. One of them is a latent feature that represents the user's preference degree to each movie genre. It is extracted by using the method proposed by Cai et al. (2014). The other two extra inputs are user's average rating and movie's average rating.

Consequently, an improved model trained on three more features is built. In order to answer $SQ3$, it needs to be examined whether and to what extent the improved model can outperform the base model proposed before. Interestingly, the results show that adding these three extra features does not help improve the model's performance. This may be caused by overfitting problems, as the extra features usually also bring in more noise.

For providing more detailed answers to the research questions, the following sections are formulated. Section 2 introduces related works that also focus on the cold-start problems in RSs. Section 3 describes the framework of the proposed model and the improved model in detail. Section 4 discusses the dataset, experimental settings and environment. The experimental results are then showed in Section 5 and analyzed more specifically in Section 6. Finally, Section 7 makes a conclusion for this paper.

2. Related work

To date, researchers have investigated a variety of approaches to mitigating cold-start problems in RSs. The literature review of this thesis mainly discusses the research works that attempt to use different sources and types of data to solve cold start problems. Some of these studies only address one of the cold-start problems, either the new user cold-start problem or the new item cold-start problem. However, they are still of considerable referential importance and have provided valuable insights in this domain, such as the methods of managing users and items auxiliary information and extracting more useful data. Some other studies have achieved to ameliorate both new user and new item cold-start problems in RSs, which provide some model architectures with reference significance and numerous insightful findings.

2.1 Addressing new item cold-start problem

Several studies have revealed that the key to solving new item cold-start problem is to fully use the existing items and users' interactive behaviors with them. These interactive behaviors can be users' reviews or purchases of the existing items. Some of the studies exploited content-based (CB) method to alleviate new item cold-start problems. Given the existing items' properties (i.e.,items' contents), and users' historical behavioral data, Saveski and Mantrach (2014) utilized non-negative matrix factorization technique to factorize the item-property matrix and the item-user rating matrix to extract the association between users and each of items' properties. Based on the association extracted and new item's properties, Saveski and Mantrach (2014) claimed that the score representing how likely the new item can interest a user can be predicted. However, Yuan et al. (2016) argued that CF method can generate more accurate and diverse recommendations than CB, as CB limits users' choices (Zhu et al. 2020). Therefore, Yuan et al. (2016) created a deep learning-based CF recommender system that leverages the document embedding technique (i.e.,doc2vec) to pair (match) each new item I_{new} with the existing item I_{old} that has ratings based on the similarity. By doing this, a pair (I_{new}, I_{old}) is generated and considered as one item. Therefore, when I_{old} is recommended to a user by utilizing CF method, I_{new} will also be recommended to this user. Yuan et al. (2016) applied the proposed system in job searching domain and achieved to make recommendations for new jobs. In addition to only focusing on items' content information, Zhu et al. (2019) took active learning method into account as well. Active learning is basically selecting some users to randomly rate new items. In order to

maximize the effectiveness of active learning, Zhu et al. (2019) designed user selection criteria based on users' historical ratings and items' properties. Thus, users who meet the criteria are selected to rate new items. According to these ratings, users' previous ratings and items' properties, Zhu et al. (2019) can predict the ratings of other users on new items. Another research work inspires me is the one from Ameen et al. (2020). The authors proposed a convolutional neural network (CNN) and matrix factorization based model that recommends travel locations to travelers. To overcome new travel location cold-start problem, the authors trained a CNN on the images of new travel locations to extract latent factor representations of them. Based on user-travel location attributes extracted from historical data and new location's latent factors, Ameen et al. (2020) achieved to recommend new locations to users who are interested in them. Ameen et al. (2020) presented a method of combining image data for recommendation in a recommender system. In the movie recommender system proposed in this thesis, movie posters, as image data, may also be exploited by using the same method. Besides, the authors tuned one of the hypeparameters in their proposed model to achieve better results. Therefore, hyperparameter optimization will be taken into account in this thesis for generating more accurate recommendations.

The aforementioned studies use some side information of items, such as their properties and descriptions, to mitigate new item cold-start cases. However, they neglect auxiliary information of users. Thus, new user cold-start problem is not solved. Low, Yap, and Tee (2018), however, took side information of users and items into account to build a hybrid model using generalized matrix factorization and CNN for movie recommendations. Nevertheless, their main goal is not to ameliorate cold-start problems and they only used user ID and movie ID as inputs, which may cause poor generalization. As (Low, Yap, and Tee 2018) summarized themselves, more side information can be involved to generate more accurate recommendations.

2.2 Addressing new user cold-start problem

Compared with dealing with new item cold-start problems, handling new user cold-start cases is sometimes considered to be harder, as new items are often launched online with auxiliary information, such as some descriptions or tags, that can be used for recommendations. As for new users, Feng et al. (2021) utilized their explicit and implicit feedback data, Lika, Kolomvatsos, and Hadjiefthymiades (2014) and Safoury and Salah (2013) purely exploited users demographic attributes to generate recommendations, while Sahebi and Cohen (2011) also took external information into account. Specifically, explicit feedback data refer to users' available ratings, and implicit feedback data can be users' clicking or browsing history (Feng et al. 2021). By using Bayesian Personalized Ranking (BPR) model on implicit feedback data, Feng et al. (2021) extracted implicit features of users and items. Likewise, based on explicit feedback data, Probabilistic Matrix Factorization (PMF) is utilized to extract explicit features. Feng et al. (2021) then combined these two types of features to train the model, and hence the fused user-item matrix can be obtained to predict recommendations. However, it should be pointed out that Feng et al. (2021) assume the users with less than 20 ratings as new users in their research, which may not be meticulous, as in real world, new users are the users without any historical information. Considering that, Sahebi and Cohen (2011), Lika, Kolomvatsos, and Hadjiefthymiades (2014) and Safoury and Salah (2013) suggested to fully use users' social networks information and demographic data for recommendation generation. The basic idea of Sahebi and Cohen (2011) is to build

communities based on users external profiles and similarities between these profiles. The external profiles can be composed of users social networks information provided in other systems. Similarly, Safoury and Salah (2013) and Lika, Kolomvatsos, and Hadjiefthymiades (2014) proposed to construct a neighborhood for new users based on the similarities on their demographic data. Thus, the items that are commonly highly-rated by the neighborhood users can be recommended to new users who are in the same neighborhood.

Generally, when users register on an online platform, they often provide some basic data, such as demographic data mentioned by Safoury and Salah (2013). Furthermore, sometimes they even directly provide or select personal preferences on the platform or connect their external profiles (Sahebi and Cohen 2011). All these data can be fully used in RSs to generate more personalized recommendations. However, these basic data are usually with multiple modalities and can be very complex (Gupta and Katarya 2020). Deep learning algorithms, in this scenario, can be the most ideal tool to manage these data to generate better results due to their powerful feature representations learning ability (Gupta and Katarya 2020; Ma, Geng, and Wang 2020). Ma, Geng, and Wang (2020) utilized deep neural networks on three types of interactions between services and mashups to extract useful features, thereby recommending appropriate component services to new mashups for developers.

2.3 Addressing both new item and new user cold-start problem

Previous studies show the importance of utilizing users and items data in RSs. Whether it is observable auxiliary information or latent features extracted from the existed data, they all play a vital role in generating accurate recommendations in the cases of cold-start. Natarajan et al. (2020) suggested to use the Open Linked Data (LOD) cloud to collect new users or new items basic data, subsequently exploit these data in RSs to make recommendations. Guo, Sun, and Theng (2019), Bi, Liu, and Fan (2020) and Bahrani et al. (2020), on the other hand, suggested in their studies that the provided side information of users and items can be used in various recommendation engines. It can be used in deep neural networks for nonlinear latent representations learning (Guo, Sun, and Theng 2019; Bi, Liu, and Fan 2020). It can also be used for similarity computing. Bahrani et al. (2020) proposed to expand users and items profiles by extracting and combining users' demographic similarity and cosine similarity, items' ontological similarity and cosine similarity to deal with cold-start cases. In addition to basic side information like users demographic data, social networks information can also be leveraged in RSs to extract similarities or links between users and items in the cold-start situation (Camacho and Alves-Souza 2018; Liu et al. 2020).

3. Methods

Deep learning algorithms are applied in this project to solve a regression problem. The main framework of the proposed model is a multi-layer feedforward neural network trained on side information of users and items to predict a continuous label —users' ratings for items. It should be mentioned that MovieLens 1M dataset (Harper and Konstan 2015) is used in this thesis to build a movie recommender system, so the items mentioned throughout this section refer to the movies. More details about the dataset itself are discussed in Section 4.1. Based on the dataset, users' side information including users' online IDs, genders, ages and occupations can be used in the deep neural network for representations learning of users. Similarly, items' side information that contains items' IDs, genres, titles and released years is utilized in the deep neural network to extract useful features of items. The extracted user's and item's features are then concatenated to act as the user-item features to represent each instance. The whole process of representations learning and concatenation is completed through the input layer, embedding layer, convolutional layer, pooling layer and dense layer in the neural network. The predicted ratings then are yielded from the output layer.
The proposed model mentioned above is considered as a base model aiming to answer the main research question, $SQ1$ and $SQ2$ that are stated in Section 1. To answer $SQ3$, not only observable side information, but also 3 latent features extracted from historical rating data will be used to examine whether additional features can help improve model's performance. The extracted latent features are user's average rating, movie's average rating and user's preference degree to each movie genre (Cai et al. 2014). However, it should be noted that in new user cold-start cases, it is not feasible to obtain user's average rating and preference degree to each movie due to the lack of history records of new users. Likewise, movie's average rating cannot be gained and used while dealing with a new movie.

In summary, a base model trained only on the observable side information of users and items and an improved model trained on both observable side information and latent information are discussed in Section 3.1 and Section 3.2, respectively.

3.1 Base model

The main framework of the base model is shown in Figure 1. A multi-layer feedforward deep neural network including the input layer, embedding layer, fully connected layer, convolutional layer, pooling layer and output layer is conducted. Next, each part of the proposed model is introduced in details.

1. Inputs
Side information of users and items utilized in the base model is either categorical or textual data that cannot be directly processed in the deep neural network, as arithmetic operations cannot be performed on the values taken by categorical and textual data (Bi, Liu, and Fan 2020). Therefore, these data are needed to be encoded to numerical values before they enter into the input layer in the neural network.

Categorical data
As for categorical data, such as user's gender, age and item's genre, released year, ordinal encoding (i.e., integer encoding) is applied. In ordinal encoding, each unique

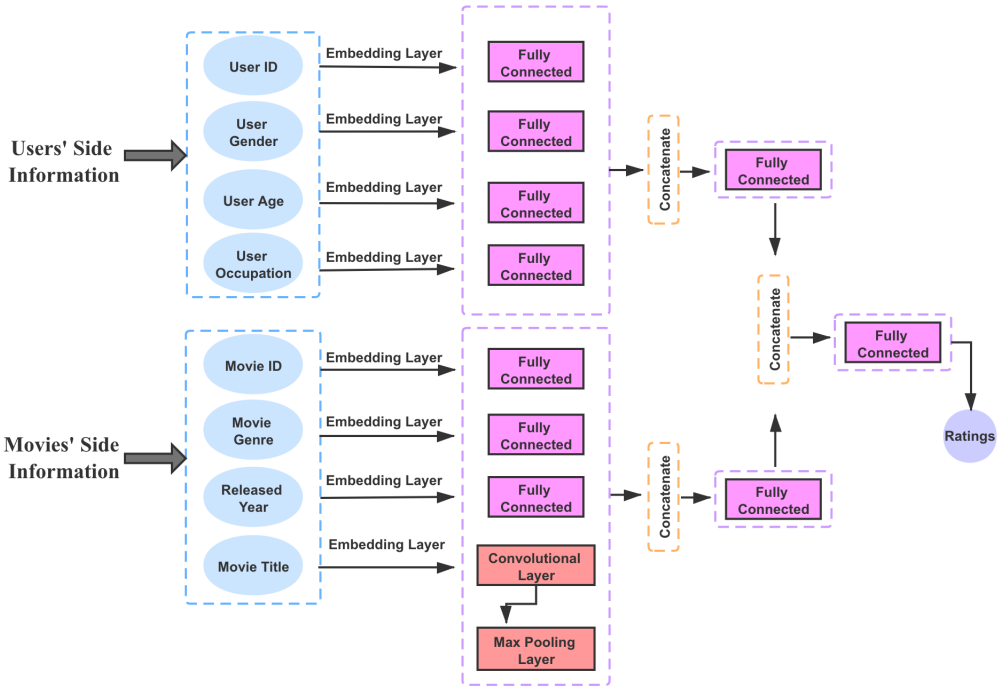

Figure 1
The main framework of the proposed model (base model) that consists of the input layer, embedding layer, fully connected layer, convolutional layer, pooling layer and output layer.

category value can be represented by an integer. The integers assigned can be arbitrary or in accordance with a certain logical relationship or order. For the categorical data mentioned above, each category value is assigned the index value of each category in the category set. For example, in the set {Animation, Comedy, Drama, Romance}, the category Animation can be represented by its index in the set, which is the number zero, and Comedy is one, Drama is two, Romance is three.

Text data
In this thesis, only item's title is considered as text data. Tokenization (i.e., word segmentation) is implemented to split the text into a sequence of strings, the element of which is called token. For example, for the movie titles: "The Cat From Outer Space", "Cats Don't Dance", and "Space Cowboys", they can be split into the lists of the words: ["The", "Cat", "From", "Outer", "Space"], ["Cats", "Don't", "Dance"], and ["Space", "Cowboys"] by using Tokenization. Each unique word in the list is a token. Each token then can be assigned a numerical value (integer) according to the frequency it appears and its position in the text. In this case, the word "Space" is assigned the number one, as it occurs the most in the text, while others occur only once. After that, the integers are endowed to other words based on their positions in the text (see Table 2). This suggests that each movie title can be represented as a sequence of integers respectively. Moreover, padding is used to ensure all text data to be with the same length, as shown in Table 3.

Table 2
An example presenting how text data (i.e., movie titles) are segmented into tokens and then converted to numerical values.

Token	Index-based Encoding
"Space"	1
"The"	2
"Cat"	3
"From"	4
"Outer"	5
"Cats"	6
"Don't"	7
"Dance"	8
"Cowboys"	9

Table 3
An example illustrating how text data (i.e., movie titles) are represented by the sequences of integers that are with the same length and can be processed in the neural network.

Text	Sequence of Integers
"The Cat From Outer Space"	[2, 3, 4, 5, 1]
"Cats Don't Dance"	[6, 7, 8]
"Space Cowboys"	[1, 9]
After Padding	
"The Cat From Outer Space"	[2, 3, 4, 5, 1]
"Cats Don't Dance"	[6, 7, 8, 0, 0]
"Space Cowboys"	[1, 9, 0, 0, 0]

However, the user and item inputs (i.e., integer representations) have very limited representation capability (Ying et al. 2018), as they are merely assigned based on the frequency and index values and independent with each other. As a result, the word "Cat" and "Cats" in the above example are completely unrelated, yet this is obviously not in line with reality. Thus, a word should be represented from different dimensions, such as the singular and plural form in this case and semantics, instead of only from the perspective of frequency and basic index.

2. Embedding layer
The embedding layer is employed to learn more informative vector representations for categorical and text data. It maps the encoded categorical and text data (i.e., integer representations) to n-dimensional dense vectors (Jun et al. 2020). Each element in the vector represents a feature or a combination of features for each discrete variable. For this reason, these vectors are generally more informative than the index-based integer representations discussed previously. It should be noted that the dimensionality of word embeddings, which is denoted by d, can be arbitrarily specified. In the base model, word embeddings are set to 32-dimensional (i.e., $d = 32$).

Take the movie title "The Cat From Outer Space" as an example, each integer representation in Table 2 can be transformed into a four-dimensional dense vector by passing in the embedding layer. The results are shown in Table 4.

3. Convolutional layer
Kim (2014) has proposed the TextCNN algorithm that leverages the convolutional neural network (CNN) to extract text features for sentence classification tasks (Gong and Ji 2018). The structure of TextCNN is mostly consistent with that of CNN when CNN is used for image representations learning (Zhang and You 2021). The only difference is that when using CNN to extract image features, since image data are

Table 4
Embedding representations endowed after passing in the embedding layer.

Token	Index-based Encoding	Four-dimensional Word Embedding
"The"	2	[-0.0400, -0.0189 , -0.0412, 0.0415]
"Cat"	3	[0.0237, -0.0418, 0.0038, -0.0208]
"From"	4	[-0.0326, -0.0395, -0.0494, -0.0398]
"Outer"	5	[0.0243, -0.0068, -0.0075, -0.026]
"Space"	1	[-0.037 , 0.0322, -0.0193, -0.0481]

two-dimensional, the convolution kernel needs to slide from the left column to the right column and from the top row to the bottom row for feature extraction, while the horizontal sliding between columns is no longer meaningful when using TextCNN to extract text features. For example, the word "Cat" in Table 4 is represented by a dense vector $[0.0237, -0.0418, 0.0038, -0.0208]$. When the convolution kernel slides from left to right, the generated vector constantly represents the same word "Cat" without gaining any other useful information. As a result, for text data, the width of the convolution kernel is the same as the dimension of word embedding d, while the height h of the convolution kernel is the number of words taken in each window (i.e., window size) (Zhang and You 2021). The height h can be adjusted generally between 3 and 5 (Zhang and You 2021) to capture various types of contextual features (Bi, Liu, and Fan 2020; Kim et al. 2017), which can interpret text data more comprehensively (Zhang and You 2021).

In summary, the TextcNN model can be considered as a variant of the CNN model with only one convolutional layer and one max pooling layer (Zhang and You 2021). In this thesis, these two layers are built after the embedding layer in the proposed model to extract item features from item titles. Let the word embedding gained from the embedding layer be denoted as e_n, where $n \in N$, representing the n_{th} word in the item title that contains N words in total. Consequently, a matrix $E^{N\times d}$ can represent an item title and a matrix with the size of $h \times d$ can represent a sliding window (convolution kernel). This indicates that there will be up to $N - h + 1$ sliding windows used to extract the contextual feature C_n of the input word embedding e_n that the submatrix $E^{h\times d}_{n:n+h-1}$ contains. Finally, the vector C is the output generated from the convolutional layer and its element C_j, where $j \in N - h + 1$, can be calculated by Equation (1):

$$C_j = f(K^{h\times d} * E^{h\times d}_{n:n+h-1} + B), \tag{1}$$

where $K^{h\times d}$ is the convolution kernel, "$*$" is the convolution operator (i.e., element-wise multiplication and sum them up), B is the bias part and f represents a non-linear activation function. Accordingly, the output vector C can be denoted by Equation (2):

$$C = [C_1, C_2, ..., C_{N-h+1}]. \tag{2}$$

Specifically, take the four-dimensional word embedding vectors that represent the first

item title "The Cat From Outer Space" presented in Table 4 as an example. Let the convolution kernel be specified as $K = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$ with the height $h = 2$ and the width $d = 4$. The bias term B is considered to be zero. Besides, the Rectified Linear Unit (ReLU, i.e., $f(z) = max(0, z)$) is set as the activation function, as it helps avoid the vanishing gradient problem (Kim et al. 2017). Thus, the process of how the convolutional layer extracts the contextual feature vector C can be shown in Figure 2. More specifically,

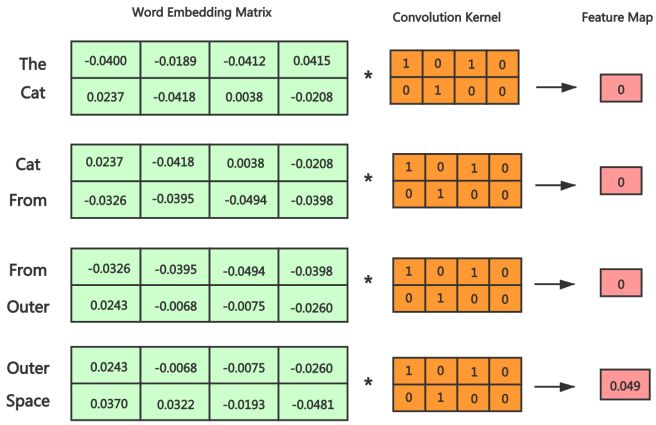

Figure 2
The input is a word embedding matrix with the size 5×4, 5 represents the number of words in this title, 4 is the dimensionality of embeddings. "$*$" denotes the convolution operator between input word embeddings and convolution kernel $K \in \mathbb{R}^{2\times4}$. The output is a vector C, representing the extracted contextual features of the title.

each element C_j in C can be calculated as below:

$$
\begin{aligned}
C_1 &= max\left(0, \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} -0.0400 & -0.0189 & -0.0412 & 0.0415 \\ 0.0237 & -0.0418 & 0.0038 & -0.0208 \end{bmatrix}\right) \\
&= max(0, (-0.0400 + 0 - 0.0412 + 0 + 0 - 0.0418 + 0 + 0)) \\
&= max(0, -0.123) \\
&= 0.
\end{aligned}
$$

Accordingly,

$$
\begin{aligned}
C_2 &= max(0, (0 + 0 + 0.0038 + 0 - 0.0326 + 0 + 0 - 0.0398)) = 0 \\
C_3 &= max(0, (0 + 0 - 0.0494 + 0 + 0.0243 + 0 + 0 - 0.026)) = 0 \\
C_4 &= max(0, (0.0243 + 0 - 0.0075 + 0 + 0 + 0.0322 + 0 + 0)) = 0.049
\end{aligned}
$$

As a result, the output of the convolutional layer can be denoted as the vector $C = [0, 0, 0, 0.049]$, which is also known as a feature map. Different convolution kernels generate different feature maps that represent the contextual features extracted from the text data by passing in the convolutional layer.

4. Max pooling layer
The max pooling layer is used to process the feature maps generated from the convolutional layer and extract the maximum values from each feature map. Such maximum values are the outputs of the max pooling layer. The advantage of using max pooling

layer is that while maintaining the important features in the feature maps, the number of parameters can be greatly decreased, thereby reducing the model's execution time. Moreover, this is also helpful to avoid overfitting to some extent. It can be seen from Figure 3 that the feature map has been transformed from four-dimensional to one-dimensional by passing in the max pooling layer.

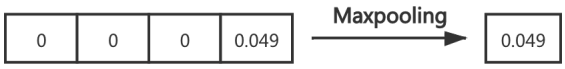

Figure 3
The four-dimensional input and one-dimensional output of the max pooling layer.

5. Dense layer and concatenation
The dense layer is also known as fully connected layer, which is used to learn the combinations of the features extracted from the previous layer. Each node of the previous layer is connected to each one of the dense layer by multiplying by a weight matrix and adding a bias term. The outputs then can be used to compute an activation function, such as ReLU, for non-linear learning. Moreover, the weights assigned represent the degree of connectivity between nodes, they are initialized to random values (usually small values like zero) and altered to reduce the errors as training process repeats (Jun et al. 2020).
As shown in Figure 1, there are three different dense layers built in the proposed model. The first dense layers are built for learning representations of each feature of the user and item (except for the item title, as representations of it is learned through the convolutional and max pooling layer). After that, all the learned representations of each user feature are concatenated to feed into the second dense layer for further learning the integrated user feature. The process of learning the item feature is same as learning the user feature. The outputs of the second dense layer are two matrices, which respectively represents the user feature and item feature. These two feature matrices are then concatenated to feed into the third dense layer to learn the integrated user-item feature. The output of the third dense layer is a matrix representing the user-item feature, which is then connected to the output layer to generate the final predictions.

3.2 Improved model

The main framework of the improved model is shown in Figure 4. As can be seen that except for the eight discrete variables used in the base model, another three continuous variables are added in the network as inputs. They are user's average rating, movie's average rating and user's preference score to each movie genre. It is worth noting that these three numerical variables are extracted based on historical rating data. User's average rating and movie's average rating can be easily computed based on the rating data. As for user's preference score to each movie category, the method of calculation proposed by Cai et al. (2014) is adopted. The idea is if the user u has rated the movie i and there are n genres of i, then each genre obtains $\frac{1}{n}$ attention score for the user u regarding to movie i. As a result, the preference score of user u for each movie genre g

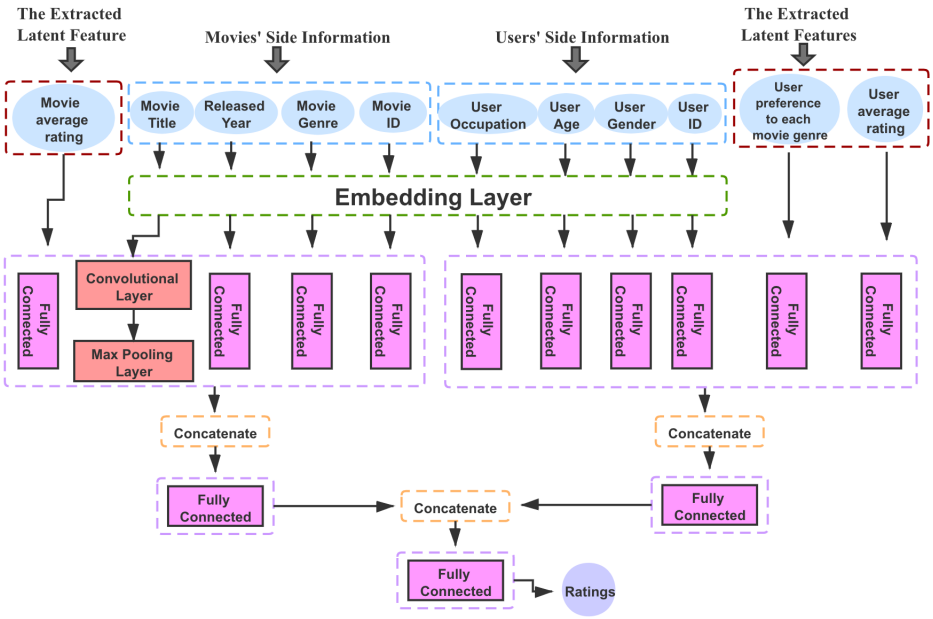

Figure 4
The main framework of the improved model. Combined with the features used in the based model, three extra numerical features are added as inputs. They are user's average rating, movie's average rating and user's preference score to each movie category.

can be computed by Equation (3) and Equation (4):

$$r_{ug} = \frac{\sum_{i \in D_k(u)} \frac{sgn(u,i,t)}{N(i)}}{k}, \tag{3}$$

$$sgn(u,i,t) = \begin{cases} 1 & \text{if genre } g \text{ belongs to movie } i \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Where, r_{ug} represents the preference degree of user u for movie genre g, $N(i)$ is the number of genres that belong to movie i, while i is one of the k movies rated by u. Accordingly, if there are j movie genres in total, then the preference degree of user u for each genre can be represented as the vector U_u shown in Equation (5)

$$U_u = (r_{u1}, r_{u2}, ..., r_{uj}). \tag{5}$$

4. Experimental setup

4.1 Dataset description

GroupLens (https://grouplens.org/) provides several classic datasets with different sizes that are specifically applicable for recommender system research. The dataset used in this project is MovieLens-1M (Harper and Konstan 2015). The dataset includes three DAT files containing users' demographic information (ID, age, gender, occupation), movies' basic data (ID, genre, title, released year), and ratings. The ratings are continuous while all the other eight variables are discrete, among which, seven are categorical (user ID, age, gender, occupation, movie ID, genre, released year) and movie titles are text data. The dataset consists of 6040 users, 3952 movies, and $1,000,209$ ratings. Each user has rated at least 20 movies (Harper and Konstan 2015).
There are a few reasons why I choose this dataset to build the model. Firstly, it provides real-world data. Besides, the dataset is large-scaled labeled. As Sun et al. (2017) demonstrate in their research, large datasets can usually enhance model's performance. Lastly, the movie titles in the dataset are identical to the ones on IMDB. Therefore, more information related to the movies on IMDB, such as movies' posters and descriptions, can be further utilized in the proposed model to achieve more accurate recommendations in the future study.

For more visually understanding the dataset and each variable, the following figures are generated. Figure 5 describes the age distribution, gender distribution, and occupation distribution of users in the dataset. As can be seen that there are seven age groups, two gender groups and 21 occupation groups in the dataset. The largest age group is young people who are around 25-34 years old, which is in line with the occupation patterns shown on the right side of the figure. Besides, there are more than twice as many male users as female users in this dataset.

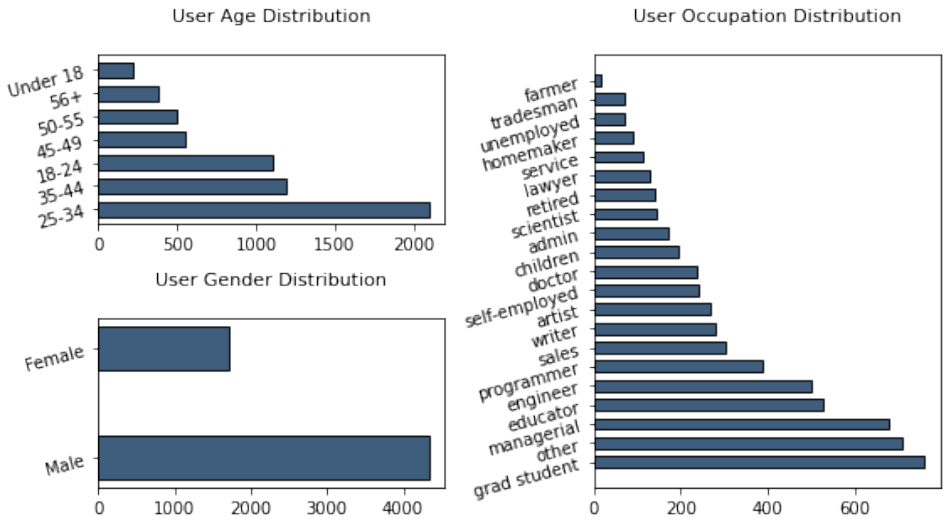

Figure 5
User data distribution in MovieLens-1M. It includes user age distribution (top left), user gender distribution (bottom left), and user occupation distribution (right side).

As for movie data, Figure 6 shows that the dataset contains 18 movie genres, of which drama movies account for nearly 50%, followed by comedy movies. By contrast, there are less than 100 film-noir movies out of 3952 movies in the dataset, which has the smallest proportion.

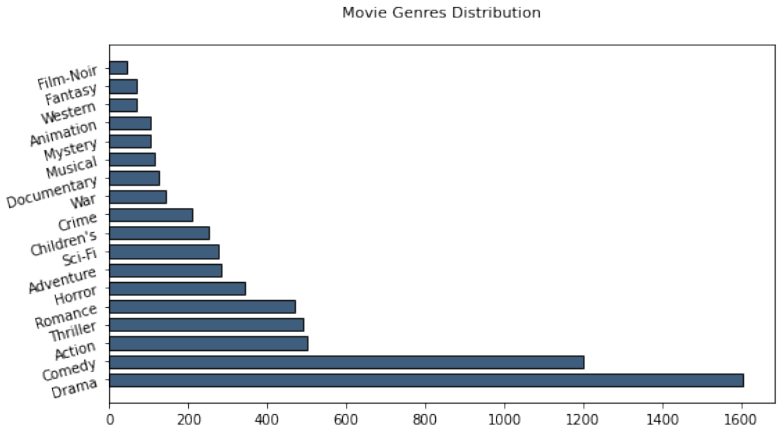

Figure 6
Movie genres distribution in MovieLens-1M. There are 18 movie genres in total, some movies have various genres. All in all, more than 40% movies are drama, less than 2.5% are film-noir.

For other types of data, such as ID, movie title, and movie released year are not plotted, as some of them only contain unique values (i.e., user ID, movie ID, and movie title), while the released year has 81 categories, so plotting may not show more clearly.
Finally, Figure 7 displays the distribution of ratings in the dataset. The ratings are on a scale of one to five and whole number only. As can be seen that most users rate movies with the scores from three to five, while only 16.4% of the total 1,000,209 ratings are between one and two.

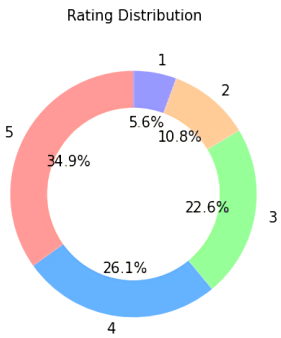

Figure 7
Ratings distribution in MovieLens-1M. There are in total 1,000,209 ratings, of which 61% are either four or five, which implies that users generally tend to give high scores.

4.2 Hyperparameter setting

Often, deep learning algorithms contain a large number of parameters, some of which can be optimized by training, such as the weight and bias terms in the neural network. Nevertheless, some of these parameters cannot be optimized by training, such as the learning rate and the number of epochs, which are known as hyperparameters. It should be noted that during the training process, the parameters like the weights are updated and the hyperparameters are constant.

Hyperparameters are different parameter values used to control the learning process, which can be tuned to influence the performance of the machine learning and deep learning models (Shankar et al. 2020 ; Kimura et al. 2020). However, when the data is as large as or even larger than the one used in this thesis, the tuning process can be extremely time-consuming (Joy et al. 2016). Considering this, only learning rate, dropout rate and batch size, as three important hyperparameters in the proposed model, are tuned based on the value of the loss function on the validation set during training process. That is, the combination of hyperparameters that can minimize the loss function on validation set will finally be used in the proposed model. In this thesis, the library Hyperopt (Bergstra et al. 2013) in Python based on Bayesian optimisation method is used for hyperparameter tuning. 80% data are randomly sampled from the original dataset for training, and 10% compose the validation set for hyperparameter optimization, the rest 10% are used for testing. The predefined loss function in this scenario is the Mean Absolute Error(MAE), which will be explained in more details in Section 4.4. The following part of this subsection mainly describes the three hyperparameters tuned in experiments, which are learning rate, dropout rate and batch size.

1. Learning rate
The learning rate is one of the most important hyperparameters affecting model's performance. It can determine how fast the model converges. In general, the learning rate is assigned to small values between zero and one. If it is set too high, the model can converge very fast, which possibly results in missing the optimal solution, whereas a small learning rate makes the model learn very slowly due to the tiny update step for the weights in the model. Therefore, an appropriate learning rate plays a vital role in improving the model's performance.

2. Dropout rate
Dropout is a regularization technique used in the neural network to alleviate overfitting problem (Krizhevsky, Sutskever, and Hinton 2012). Krizhevsky, Sutskever, and Hinton (2012) have proposed that only keep half of the neurons active while training to improve the generalization ability of the network. That is, each neuron can be set to zero (i.e., dropped out) with a probability of 0.5, thereby ensuring that the presence of a neuron does not depend on other specific ones. Thus, the neural network can be more robust, as it learns the representations that are in conjunction with various random sets of neurons, instead of only the same set of neurons (Krizhevsky, Sutskever, and Hinton 2012). Therefore, in addition to the learning rate, dropout rate, which is the probability of each neuron being set to zero can also greatly influence model's performance. In this thesis, dropout is used on the last dense layer before the output layer to prevent the model from overfitting.

3. Batch size
As noted by Bottou, Curtis, and Nocedal (2018), deep learning optimization is based on the optimization algorithms, such as Gradient Descent and its variants. As one of the variants, Mini-batch Gradient Descent is a usual choice, as it goes through a stochastic subset of training examples in each iteration, rather than going over the whole training set (Batch Gradient Descent), thereby increasing the efficiency of computing gradients. Its update rule relies on the gradient of the loss function on the training set, based on a stochastic subset of training examples (Masters and Luschi 2018). Batch size, therefore, represents the size of this small subset (mini-batch) that will be propagated through the neural network. More specifically, a large batch size can reduce training time, as with the same number of epochs, a large batch size leads to a reduction in the number of batches. However, a large batch size may also impair the generalization performance of the model due to insufficient training time and fewer updates for the parameters under the same number of epochs (Hoffer, Hubara, and Soudry 2017). Hence, tuning batch size to be smaller may contribute to improving the model's generalization performance (Masters and Luschi 2018).

In summary, Table 5 shows the search space configuration of the hyperparameters for the proposed model, where the learning rate has four possible values exponentially ranging between 10^{-2} and 10^{-5}, while the dropout rate has ten potential values within one and the batch size value is among 65, 115, 165, 215 and 265. In fact, there should be 180 possible combinations in total if use grid search method. However, only 30 combinations (trials) are tested in this thesis by applying Bayesian optimisation method.

Table 5
The search space configuration of the hyperparameters.

Hyperparameter	Range
Learning rate	10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}
Dropout rate	0 - 0.9, step = 0.1
Batch size	65 - 265, step = 50

Except for the tuned hyperparameters mentioned above, other hyperparameters involved in the model are set to arbitrary values. Section 3.1 has noted that the dimension of word embeddings is 32 (i.e., the width of the convolution kernel $d = 32$). The height of the convolution kernel, namely the window size h is set to the integers between two and five for each trial. The number of such convolution kernel used in the convolutional layer is eight. As for dense layer, there are three dense layers with different output dimensions built in the proposed model. The output dimension of the first, second and third dense layer is 32, 200 and 64, respectively. Besides, the model is trained for 50 epochs with the early stopping technique applied for addressing overfitting issues.

4.3 Baselines and competitors

In experiments, the proposed model is firstly compared with two baselines, including:

- User average rating (UserMean): the mean of the target user's available ratings on every movie is used as the predicted rating for each user.
- Movie average rating (MovieMean): the mean of the target movie's available ratings from every user is used as the predicted rating for each movie.

These two baselines are selected due to two reasons. Firstly, they are straightforward and easy to compute and understand. Secondly, the other compared algorithms used in the experiments are not capable of dealing with the cold-start problems, while the movie average rating can be a baseline when dealing with the new user cold-start problem in the system and the user average rating can be a baseline in the new item cold-start situation.

Apart from the basic baselines, three classic recommender system algorithms are adopted as competitors to verify the effectiveness of the proposed model, those are:

- User based Collaborative Filtering (UBCF): it computes the similarities between each user to find the users who have similar preferences, thereby recommending the items that these similar users like (Breese, Heckerman, and Kadie 1998).
- Item based Collaborative Filtering (IBCF): it computes item-item similarities to find the items with similar rating patterns, thereby generating recommendations based on these patterns (Sarwar et al. 2001). Besides, Cosine similarity is used as the similarity measure in both user based and item based collaborative filtering methods.
- Singular Value Decomposition (SVD): it is a method of matrix factorization (Sarwar et al. 2000). Utilizing this function, the missing ratings can be predicted by decomposing the user-item rating matrix. As shown in Equation (6), the sparse user-item rating matrix $A \in \mathbb{R}^{m \times n}$ (m represents the number of users and n denotes the number of items) can be decomposed into three matrices:

$$A = U \Sigma V^T, \tag{6}$$

where, $U \in \mathbb{R}^{m \times k}$ is the user feature matrix, representing user's preference for k features, while $V \in \mathbb{R}^{n \times k}$ performs as the item feature matrix, indicating the relevance of each item with k features. Besides, $\Sigma \in \mathbb{R}^{k \times k}$ is a diagonal matrix with positive values in descending order as the diagonal entries (i.e., singular values). These values can be considered as the eigenvalues that determine the movement of the matrix A towards the direction of the corresponding eigenvectors.

4.4 Evaluation metrics

Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) metrics are used to evaluate the performance of the proposed model and the compared algorithms. MAE is the mean of the absolute errors between predicted values and actual values,

while RMSE is the square root of the mean squared errors between predicted and actual values. Both of the metrics are typically used in RSs for evaluation. They can be calculated by Equation (7) and Equation (8):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|\,, \tag{7}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}, \tag{8}$$

where n is sample size, $\hat{y}_i$ is the predicted rating and y_i is the actual rating.

4.5 Software

The experiments are conducted by using Python (version of 3.7.10) programming language. Several main libraries in Python Provide different technical supports for each part of the experiments. The library Keras (Chollet et al. 2015) using Tensorflow as backend contributes to building the whole neural network. The library Hyperopt (Bergstra et al. 2013) mentioned in Section 4.2 is used for hyperparameter optimization. Besides, as a Python scikit, Surprise (Hug 2020) provides the ready-to-use competitor algorithms discussed in Section 4.3. Moreover, the whole modeling process is conducted on Google Colab platform with the use of graphics processing unit (GPU).

5. Results

This section contains three subsections. Section 5.1 mainly analyzes the results of hyperparameter optimization. Section 5.2 reports the results of comparing the proposed model (base model) with the baselines and competitors mentioned in Section 4.3. Section 5.3 compares the performance of the base model and the improved model.

5.1 Hyperparameters

The first sub research question focuses on the impact of hyperparameters on model's performance. According to Section 4.2, 30 combinations of three hyperparameters (i.e., learning rate, dropout rate and batch size) are used for hyperparameter optimization. The validation loss generated per iteration is presented in Figure 8. More specific results obtained from hyperparameter tuning process are set out in Appendix A. It can be seen that the loss hovers between 0.67 and 0.78, which indicates changes in hyperparameters do affect the performance of the proposed model. The minimum loss occurs in the fifth iteration, which is about 0.67, while the maximum loss appears in the seventh iteration, with the value around 0.78. Besides, based on the results, the relation between each hyperparameter (i.e., learning rate, dropout rate, batch size) and the resulted validation loss in 30 iterations will be discussed as following.

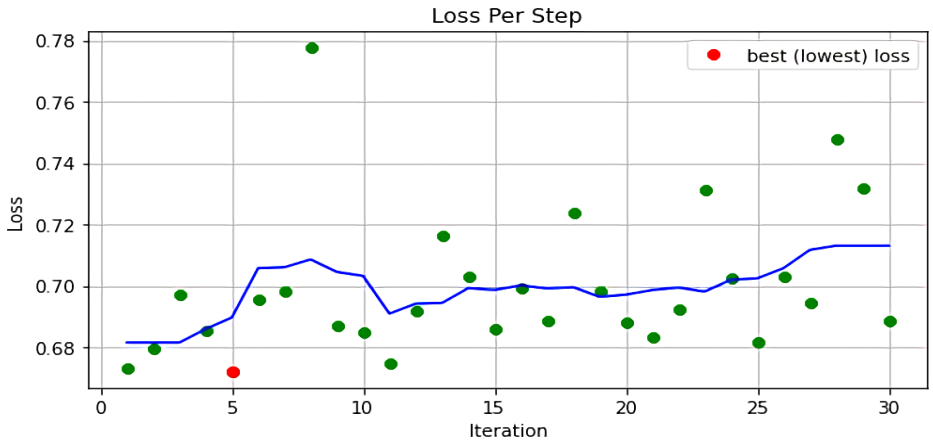

Figure 8
The overall hyperparameter tuning results in 30 iterations.

1. Learning rate
Figure 9 displays the results of tuning learning rate in 30 iterations. It shows that the best result occurs when the learning rate is set to 10^{-4}, where the resulted validation losses are quite steady, ranging approximately between 0.67 and 0.69 constantly. Another

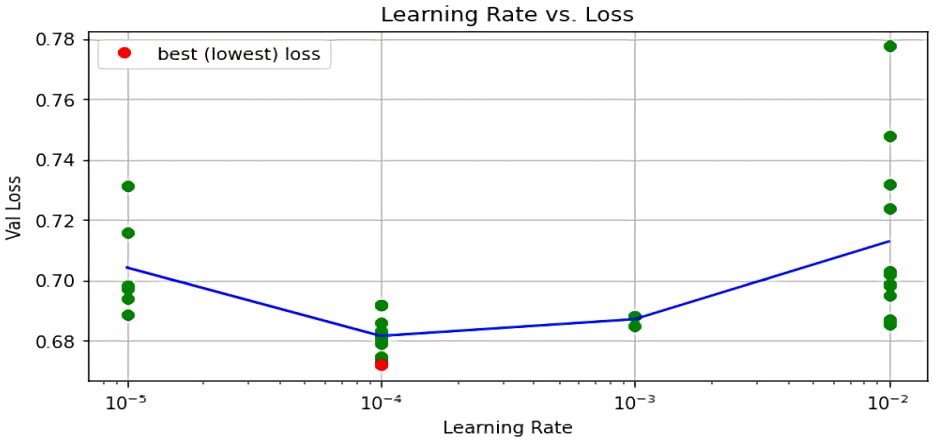

Figure 9
The results of adjusting learning rate in 30 iterations.

observation from Figure 9 is the learning rate is positively correlated with validation loss in general (a figure showing the correlation between each hyperparameter and validation loss can be seen in Appendix B).

2. Dropout rate
The results of tuning dropout rate in 30 iterations are presented in Figure 10. It can be seen that the best result emerges when the dropout rate is set to 0.2. The correlation between dropout rate and validation loss in this case is not very obvious.

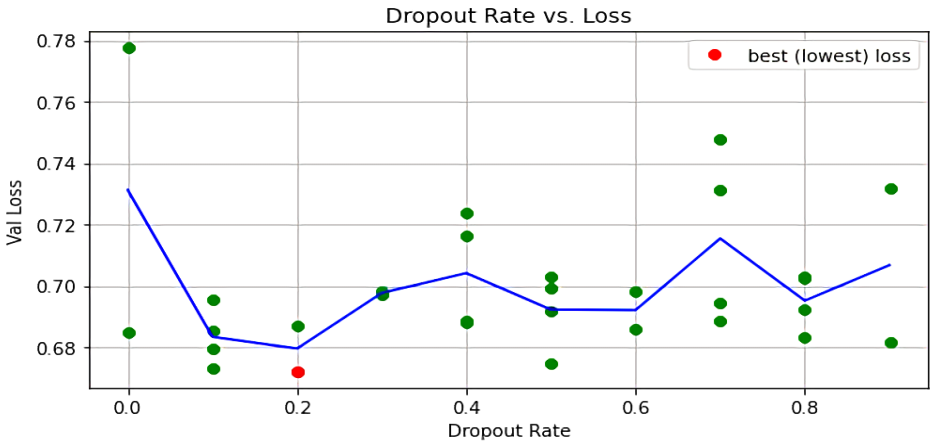

Figure 10
The results of tuning dropout rate in 30 iterations.

3. Batch size
As for batch size, Figure 11 illustrates that the best result occurs when the batch size is configured to 64, which is the smallest value in the search space configuration of the batch size. This result is in line with the conclusion made by Masters and Luschi (2018), who argue that a small batch size tends to improve the model's performance.

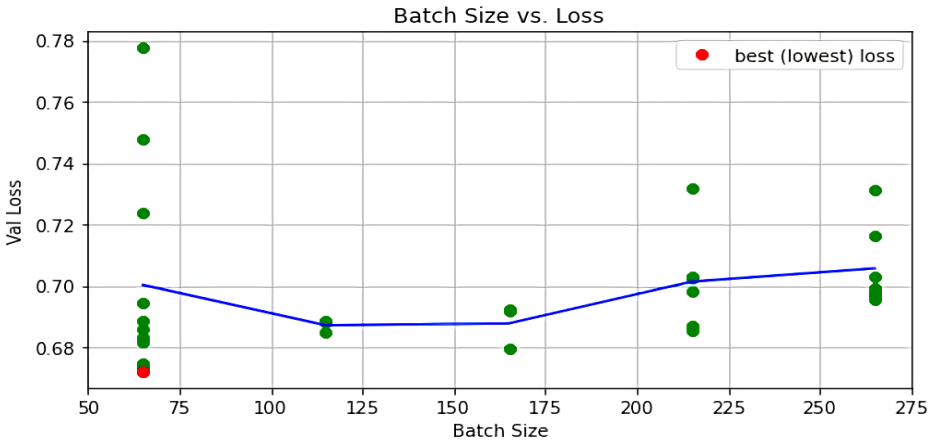

Figure 11
The results of tuning batch size in 30 iterations.

5.2 Base model vs. baselines and competitors

There are two proposed models in the thesis. One is the base model only utilizing users and movies observable basic data to predict the ratings, while the other one is considered to be an improved model with extra latent features added in the neural

network to predict the ratings. In this subsection, the proposed model refers to the base model. Moreover, for each algorithm, 60%, 70%, 80%, 90% data are randomly sampled from original datasetas for training, and the rest are used for testing. The reported results are testing scores.

The second sub research question explores if the proposed model outperform other classic recommender system algorithms, which in this thesis are user-based CF, item-based CF and SVD approaches. Except for these algorithms, two simple baseline methods are also compared with the proposed model. All the results of comparisons are set out in Table 6, based on these results, a few insightful observations are gained and explained as following.

Table 6
The performance comparison among UserMean, MovieMean, User-based CF, Item-based CF, SVD, and the proposed DNN model on MovieLens-1M dataset.

		Evaluation Metric	
Setting	**Models**	**MAE**	**RMSE**
60%	UserMean	0.8236	1.0287
	MovieMean	0.7779	0.9738
	User-based CF	0.7759	0.9806
	Item-based CF	0.7899	1.0058
	SVD	0.7023	0.8931
	DNN (the proposed model)	**0.6866**	**0.8929**
70%	UserMean	0.8239	1.0287
	MovieMean	0.7782	0.9743
	User-based CF	0.7736	0.9790
	Item-based CF	0.7843	1.0023
	SVD	0.6943	0.8840
	DNN (the proposed model)	**0.6819**	**0.8860**
80%	UserMean	0.8233	1.0286
	MovieMean	0.7776	0.9737
	User-based CF	0.7709	0.9768
	Item-based CF	0.7807	0.9999
	SVD	0.6864	0.8750
	DNN (the proposed model)	**0.6728**	**0.8750**
90%	UserMean	0.8222	1.0278
	MovieMean	0.7753	0.9717
	User-based CF	0.7684	0.9759
	Item-based CF	0.7772	0.9972
	SVD	0.6805	0.8678
	DNN (the proposed model)	**0.6690**	**0.8717**

1. Baselines
In the case of new user cold-start, MovieMean can be used as new user's possible rating for each movie to generate recommendations. Likewise, UserMean can be used while

dealing with new item cold-start situation. However, not surprisingly, simply taking user's or movie's average rating as the predicted value is not reliable, as it assumes users do not have any preference. From the results, UserMean and MovieMean perform worse than the proposed model constantly for each training setting, in terms of both MAE and RMSE metrics. However, both UserMean and MovieMean methods are relatively time-saving. Given the same computation time, MovieMean performs slightly better than UserMean in this case.
The results indicate that the proposed model performs significantly better than the baselines. On the other hand, the proposed model is also able to alleviate both new user and new item cold start problems because it does not rely on any historical information, but only trains a network based on the auxiliary information of users and items to fully learn the features of them, thereby predicting the corresponding ratings. Thus, the proposed model can be believed to be a relatively effective method to deal with cold-start problems in RSs.

2. Competitors
The proposed model outperforms all the competitors for each sampling setting in terms of MAE metric. This implies that by leveraging users and movies basic data, neural networks can learn more complex features, which is a strong support for prediction. However, for each sampling setting, SVD slightly outperforms or equals to the proposed model in terms of RMSE metric, even though the differences are very small, which are 0.0007, 0.002, and 0.0039.

3. Training size
Another observation that meets expectations is that all models except UserMean and MovieMean perform better as the size of training set increases. A larger dataset can provide more data for training, which can make the model perform better. This finding is consistent with the research conclusion made by Sun et al. (2017).

5.3 Base model vs. improved model

The third sub research question is proposed to understand whether adding extra features can improve model's performance. The extra features used in the improved model are user's average rating, movie's average rating and user's preference distribution to each movies genre. By using the same hyperparameter settings reported in Section 5.1, each model is trained on 60%, 70%, 80%, and 90% data, respectively, while the rest are used for testing.

Table 7 shows the results of comparing the improved model with the base model. After adding three more extra features, the improved model performs basically the same as the base model. For 60% and 70% sampling settings, the base model performs slightly better than the improved model, in terms of both MAE and RMSE. However, when training size increases to 80% and 90%, the improved model barely outperforms the base model: MAE is 0.0009 less than the base model, while RMSE still 0.001 higher than the base model for 80% training size; MAE is 0.00036 less than the base model, RMSE is 0.0041 lower than the base model for 90% training size. In addition, as the training data increases, both models perform better.

Table 7
The performance comparison between base model and improved model on MovieLens-1M dataset. The base model utilizes eight features that are observable basic data of users and movies. Based on that, the improved model adds three more features that are latent and extracted from the present data.

		Evaluation Metric	
Setting	**Models**	**Testing MAE**	**Testing RMSE**
60%	Base model	0.6866	0.8929
	Improved model	**0.6891**	**0.8943**
70%	Base model	0.6819	0.8860
	Improved model	**0.6830**	**0.8884**
80%	Base model	0.6728	0.8750
	Improved model	**0.6719**	**0.8760**
90%	Base model	0.6690	0.8717
	Improved model	**0.6654**	**0.8676**

6. Discussion

The main goal of this thesis is to explore the possibility of utilizing users and items side information in deep neural networks to mitigate cold-start problems that plague RSs. After a neural network trained on users and items auxiliary information is built, its performance needs to be optimized and evaluated. Thus, the first and second sub-questions are generated. Furthermore, since Bi, Liu, and Fan (2020) and Feng et al. (2021) suggested to also take user-item interaction data (i.e.,historical rating records) into consideration, the third sub-question in this thesis then explores whether extra information extracted from historical records can enhance model's performance. From another perspective, as historical rating data are not available in cold-start cases, the third sub-question can also be understood as: in order to mitigate cold-start problems, whether purely using side information as inputs while ignoring the use of historical data in the proposed model will compromise model's performance.

The classic MovieLens-1M dataset that contains plenty observable side information of users and items is used to train and evaluate the proposed model. Considering the complexity of the dimensionality and modality of these side information data, deep learning algorithms are adopted to manage the data due to their superiority on feature representations learning (Zhang et al. 2018; Gupta and Katarya 2020; Ma, Geng, and Wang 2020). Compared with traditional CB and CF approaches, deep learning algorithms usually have various hyperparameters that can influence model's performance. Kim et al. (2017), Zhu et al. (2019) and Ameen et al. (2020) implemented hyperparameter optimization in their recommendation engines. The results indicate that different hyperparameter settings can greatly impact the accuracy of recommendations. In this thesis, three hyperparameters (i.e., learning rate, dropout rate, and batch size) are tuned to find the optimal model. The validation loss during tuning hovers between 0.67 and 0.78, which confirmed that hyperparameters have

a great impact on model's performance. Consequently, the model that generates the lowest validation loss (i.e., 0.67) is selected as the final proposed model.

After model selection, the second sub research question further investigates the effectiveness of the proposed model by comparing it with baselines and the state-of-the-art algorithms. Much of the literature discussed in Section 2 has emphasized and confirmed that the use of users and items side information in RSs can alleviate cold-start problems. Moreover, the models trained on either items side information or users side information can outperform other traditional RSs that are based merely on historical information. Low, Yap, and Tee (2018), especially concluded in their research that more side information should be utilized in the proposed model to improve its performance. Inspired by these, this research utilize all observable side information of users (i.e., ID, gender, age, occupation) and movies (i.e., ID, genre, title, released year) presented in MovieLens-1M to train the neural networks. The results show that the proposed model outperforms all the other baselines and competitors in terms of MAE metric, which is in line with other literature's expectations for the full use of auxiliary information in RSs. However, there is a surprising observation about the competitors. Item-based CF performs worse than the baseline MovieMean for each sampling setting, even though as the training size increases, the performance of item-based CF becomes better.

It is worth noting that none of the studies mentioned in literature review has purely used both users and items observable side information in their models. This implies that the proposed model in this thesis may be considered as a novel approach to dealing with both new user and new item cold-start problems in RSs. On the other hand, this fact may also cause limitations for the proposed model. Firstly, users side information can be invalid, as some users may not provide real or sufficient information due to data sensitivity and privacy, which can greatly impair the accuracy of recommendations. However, there are various data sources can be used to collect more new users and new items side information, such as the LOD Cloud suggested by Natarajan et al. (2020) and social networks suggested by Camacho and Alves-Souza (2018) and Liu et al. (2020), but this must be implemented within the scope of the law. Besides, for some internet-based platforms without enough users and items data for training, the generalization ability of the proposed model will be limited.

The third sub research question attempts to examine the hypothesis whether adding some features that are extracted from historical rating data can generate more accurate recommendations. In this research, three extra features (i.e., user average rating, movie average rating, user preference degree for each movie genre) are added in the network as inputs to train together with the previous 8 features. The results revealed that only when training size reaches 80% and 90%, the improved model with 3 extra features can perform slightly better than the base model trained merely on side information. However, the performance difference between two models is negligible. Therefore, the latent features extracted from historical data do not have much effect on model's performance, which is not consistent with the expectations and insights of Bi, Liu, and Fan (2020) and Feng et al. (2021). Nevertheless, more latent features extracted by some powerful algorithms, such as matrix factorization, should still be added in the proposed model to further investigate the improvement on model's performance.

In summary, the proposed model can be used in cold-start cases, as it does not rely on any historical rating data. Furthermore, based on the experimental results, the proposed model can generate more accurate recommendations compared with the mainstream approaches, such as CF and SVD algorithms. However, the experiments and the proposed model itself are not without their downsides. As mentioned above, there may not necessarily be sufficient data and valid users side information to train the model. Secondly, integer encoding is applied to convert categorical data to numeric values, which may cause an unfair bias between categories in high dimensional space in terms of distance metrics. To alleviate this potential issue, one-hot encoding can be considered and used to pre-process the categorical data in the future study. Besides, in this research, only two trials are conducted for each sampling setting, and the best results are chosen for comparisons. However, more independent trials are needed for training, and the average score of them should be calculated, thereby generating more valid results. This was conducted by Zhuang et al. (2017) and should be taken into account in the future study. Moreover, in this study, only three hyperparameters are optimized due to the limitations of time and experimental hardware conditions. In the future, more hyperparameters, such as the number of filters in convolutional layer and the number of epochs, should be tuned to achieve the optimal model. Finally, the model is based on deep neural networks, which are often criticized to be incomprehensible like a black box due to their complex internal structures (He, Ma, and Wang 2020). Fortunately, abounding studies have proposed various methods to ameliorate this limitation (Liang et al. 2021).

7. Conclusion

This thesis attempts to explore a deep neural networks based method that can alleviate cold-start problems in RSs by using users and items side information. MovieLens-1M, as a classic dataset for researching RSs, is adopted to train and evaluate the proposed model's performance. Therefore, the recommender system built in this thesis is a movie recommender system. However, the proposed model can be widely applied to various fields, such as job search and e-commerce platforms, as long as users and items auxiliary information can be provided.

As stated in the Introduction section (Section 1), the main research question that the experiments are conducted to solve is: *"How does the deep neural network leverage users' and items' side information to alleviate cold-start problems in recommender systems(RSs)?"* To answer this question, a deep neural network trained merely on side information of users and items is proposed as a base model. The model does not use any historical data as input, hence, it can be applied in cold-start cases. To examine the validity of the model and further answer the main research question, the following three sub questions are formulated as below:

Sub question 1: *"How hyperparameters, such as learning rate, dropout rate, and batch size, affect the accuracy of the proposed model?"*

To answer this question, the optimization of three hyperparameters is done. The results show that there is a strong positive correlation between learning rate and validation loss. As for dropout rate and batch size, they have no apparent correlation with validation loss, but this may be caused by insufficient trials. In the end, a model with

relatively small learning rate, dropout rate and batch size is selected, as it yields the least validation loss. However, there are many other hyperparameters in the model, which should also be tuned in order to get better results. This can be done in the future study if time and experimental hardware conditions allow.

Sub question 2: *"Does the proposed model outperform other classic RSs models?"*

After answering sub question 1, the optimal model is generated and then compared with several baselines and competitors to evaluate the model's performance. The results show that the proposed model outperforms all the baselines (i.e., user's average rating and item's average rating) and competitors (i.e, user based CF, item-based CF, and SVD) in terms of MAE metric. This implies that compared with the mainstream CF and SVD algorithms, the proposed model can be used in practical to generate more accurate recommendations. However, as the proposed model is only trained on side information of users and items, once there is not sufficient side information provided or there is unreal side information existing, the performance of the model will then be greatly reduced.

Sub question 3: *"Instead of only utilizing observable side information, does adding extra latent features improve the proposed model's performance?"*

The main goal of this thesis is to explore an approach to mitigating both new user and new item cold-start problems, thus, the proposed model does not use any features extracted from historical data. As suggested by several studies, three extra latent features (i.e., user's average rating, item's average rating, and user's preference degree) extracted from historical data are added in the neural network as inputs to further improve the performance of the proposed model. On the other hand, this can also investigate whether only use side information in the proposed model to solve the cold start problem will be at the expense of the accuracy of recommendations. The results indicate that the impact of the extra latent features on the performance of the model is next to nothing. However, in this thesis, only three extra features simply extracted from historical data are exploited in the proposed model. In the future study, different approaches such as matrix factorization, can be utilized to extract more latent information, which may be combined into the proposed model to generate more accurate recommendations.

References

Ameen, Thaair, Ling Chen, Zhenxing Xu, Dandan Lyu, and Hongyu Shi. 2020. A convolutional neural network and matrix factorization-based travel location recommendation method using community-contributed geotagged photos. *ISPRS International Journal of Geo-Information*, 9(8):464.

Bahrani, Payam, Behrouz Minaei-Bidgoli, Hamid Parvin, Mitra Mirzarezaee, Ahmad Keshavarz, and Hamid Alinejad-Rokny. 2020. User and item profile expansion for dealing with cold start problem. *Journal of Intelligent & Fuzzy Systems*, (Preprint):1–13.

Barjasteh, Iman, Rana Forsati, Dennis Ross, Abdol-Hossein Esfahanian, and Hayder Radha. 2016. Cold-start recommendation with provable guarantees: A decoupled approach. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1462–1474.

Bergstra, James, Dan Yamins, David D Cox, et al. 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, volume 13, page 20, Citeseer.

Bi, Jian-Wu, Yang Liu, and Zhi-Ping Fan. 2020. A deep neural networks based recommendation algorithm using user and item basic data. *International Journal of Machine Learning and Cybernetics*, 11(4):763–777.

Bottou, Léon, Frank E Curtis, and Jorge Nocedal. 2018. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311.

Breese, John S, David Heckerman, and Carl Kadie. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52.

Cai, Guoyong, Rui Lv, Hao Wu, and Xia Hu. 2014. An improved collaborative method for recommendation and rating prediction. In *2014 IEEE International Conference on Data Mining Workshop*, pages 781–788, IEEE.

Camacho, Lesly Alejandra Gonzalez and Solange Nice Alves-Souza. 2018. Social network data to alleviate cold-start in recommender system: A systematic review. *Information Processing & Management*, 54(4):529–544.

Chollet, François et al. 2015. keras.

Da'u, Aminu and Naomie Salim. 2019. Recommendation system based on deep learning methods: a systematic review and new directions. *Artificial Intelligence Review*, pages 1–40.

Feng, Junmei, Zhaoqiang Xia, Xiaoyi Feng, and Jinye Peng. 2021. Rbpr: A hybrid model for the new user cold start problem in recommender systems. *Knowledge-Based Systems*, 214:106732.

Gong, Linyuan and Ruyi Ji. 2018. What does a textcnn learn? *arXiv preprint arXiv:1801.06287*.

Guo, Qing, Zhu Sun, and Yin-Leng Theng. 2019. Exploiting side information for recommendation. In *International Conference on Web Engineering*, pages 569–573, Springer.

Gupta, Garima and Rahul Katarya. 2020. Research on understanding the effect of deep learning on user preferences. *Arabian Journal for Science and Engineering*, pages 1–40.

Harper, F Maxwell and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19.

He, Congjie, Meng Ma, and Ping Wang. 2020. Extract interpretability-accuracy balanced rules from artificial neural networks: A review. *Neurocomputing*, 387:346–358.

Hoffer, Elad, Itay Hubara, and Daniel Soudry. 2017. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *arXiv preprint arXiv:1705.08741*.

Hug, Nicolas. 2020. Surprise: A python library for recommender systems. *Journal of Open Source Software*, 5(52):2174.

Joy, Tinu Theckel, Santu Rana, Sunil Gupta, and Svetha Venkatesh. 2016. Hyperparameter tuning for big data using bayesian optimisation. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2574–2579, IEEE.

Jun, Han Jong, Jae Hee Kim, Deuk Young Rhee, and Sun Woo Chang. 2020. "seoulhouse2vec": An embedding-based collaborative filtering housing recommender system for analyzing housing preference. *Sustainability*, 12(17):6964.

Kim, Donghyun, Chanyoung Park, Jinoh Oh, and Hwanjo Yu. 2017. Deep hybrid recommender systems via exploiting document context and statistics of items. *Information Sciences*, 417:72–87.

Kim, Yoon. 2014. Convolutional neural networks for sentence classification.

Kimura, Gabriela Y, Diego R Lucio, Alceu S Britto Jr, and David Menotti. 2020. Cnn hyperparameter tuning applied to iris liveness detection. *arXiv preprint arXiv:2003.00833*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105.

Liang, Yu, Siguang Li, Chungang Yan, Maozhen Li, and Changjun Jiang. 2021. Explaining the black-box model: A survey of local interpretation methods for deep neural networks. *Neurocomputing*, 419:168–182.

Lika, Blerina, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. 2014. Facing the cold start problem in recommender systems. *Expert Systems with Applications*, 41(4):2065–2073.

Liu, Siwei, Iadh Ounis, Craig Macdonald, and Zaiqiao Meng. 2020. A heterogeneous graph neural model for cold-start recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2029–2032.

Low, Yat Hong, Wun-She Yap, and Yee Kai Tee. 2018. Convolutional neural network-based collaborative filtering for recommendation systems. In *International Conference on Robot Intelligence Technology and Applications*, pages 117–131, Springer.

Ma, Yutao, Xiao Geng, and Jian Wang. 2020. A deep neural network with multiplex interactions for cold-start service recommendation. *IEEE Transactions on Engineering Management*, 68(1):105–119.

Masters, Dominic and Carlo Luschi. 2018. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*.

Natarajan, Senthilselvan, Subramaniyaswamy Vairavasundaram, Sivaramakrishnan Natarajan, and Amir H Gandomi. 2020. Resolving data sparsity and cold start problem in collaborative filtering recommender system using linked open data. *Expert Systems with Applications*, 149:113248.

Safoury, Laila and Akram Salah. 2013. Exploiting user demographic attributes for solving cold-start problem in recommender system. *Lecture Notes on Software Engineering*, 1(3):303–307.

Sahebi, Shaghayegh and William W Cohen. 2011. Community-based recommendations: a solution to the cold start problem. In *Workshop on recommender systems and the social web, RSWEB*, page 60.

Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl. 2000. Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science.

Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295.

Saveski, Martin and Amin Mantrach. 2014. Item cold-start recommendations: learning local collective embeddings. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 89–96.

Shankar, K, Yizhuo Zhang, Yiwei Liu, Ling Wu, and Chi-Hua Chen. 2020. Hyperparameter tuning deep learning for diabetic retinopathy fundus image classification. *IEEE Access*, 8:118164–118173.

Son, Le Hoang. 2016. Dealing with the new user cold-start problem in recommender systems: A comparative review. *Information Systems*, 58:87–104.

Sun, Chen, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. 2017. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852.

Wang, Ruiqin, Yunliang Jiang, and Jungang Lou. 2020. Tdr: Two-stage deep recommendation model based on msda and dnn. *Expert Systems with Applications*, 145:113116.

Ying, Haochao, Fuzhen Zhuang, Fuzheng Zhang, Yanchi Liu, Guandong Xu, Xing Xie, Hui Xiong, and Jian Wu. 2018. Sequential recommender system based on hierarchical attention network. In *IJCAI International Joint Conference on Artificial Intelligence*.

Yuan, Jianbo, Walid Shalaby, Mohammed Korayem, David Lin, Khalifeh AlJadda, and Jiebo Luo. 2016. Solving cold-start problem in large-scale recommendation engines: A deep learning approach. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1901–1910, IEEE.

Zhang, Libo, Tiejian Luo, Fei Zhang, and Yanjun Wu. 2018. A recommendation model based on deep neural network. *IEEE Access*, 6:9454–9463.

Zhang, Tianyu and Fucheng You. 2021. Research on short text classification based on textcnn. In *Journal of Physics: Conference Series*, volume 1757, page 012092, IOP Publishing.

Zhu, Xiaosong, Jingfeng Guo, Shuang Li, and Tong Hao. 2020. Facing cold-start: A live tv recommender system based on neural networks. *IEEE Access*, 8:131286–131298.

Zhu, Yu, Jinghao Lin, Shibi He, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. 2019. Addressing the item cold-start problem by attribute-driven active learning. *IEEE Transactions on Knowledge and Data Engineering*, 32(4):631–644.
Zhuang, Fuzhen, Zhiqiang Zhang, Mingda Qian, Chuan Shi, Xing Xie, and Qing He. 2017. Representation learning via dual-autoencoder for recommendation. *Neural Networks*, 90:83–89.

Appendices

A. Hyperparameter tuning results

Table 8
The Search Space Configuration of The Hyperparameters.

Iteration	Loss	Learning rate	Dropout rate	batch size
1	0.6733	10^{-4}	0.1	65
2	0.6793	10^{-4}	0.1	165
3	0.6971	10^{-5}	0.3	265
4	0.6856	10^{-2}	0.1	215
5	0.6721	10^{-4}	0.2	65
6	0.6950	10^{-2}	0.1	265
7	0.6982	10^{-5}	0.3	265
8	0.7775	10^{-2}	0.0	65
9	0.6870	10^{-2}	0.2	215
10	0.6849	10^{-3}	0.0	115
11	0.6749	10^{-4}	0.5	65
12	0.6917	10^{-4}	0.5	165
13	0.7161	10^{-5}	0.4	265
14	0.7032	10^{-2}	0.5	215
15	0.6860	10^{-4}	0.6	65
16	0.6991	10^{-2}	0.5	265
17	0.6886	10^{-5}	0.4	65
18	0.7237	10^{-2}	0.4	65
19	0.6982	10^{-2}	0.6	215
20	0.6881	10^{-3}	0.4	115
21	0.6834	10^{-4}	0.8	65
22	0.6922	10^{-4}	0.8	165
23	0.7312	10^{-5}	0.7	265
24	0.7022	10^{-2}	0.8	215
25	0.6815	10^{-4}	0.9	65
26	0.7028	10^{-2}	0.8	265
27	0.6943	10^{-5}	0.7	65
28	0.7478	10^{-2}	0.7	65
29	0.7320	10^{-2}	0.9	215
30	0.6884	10^{-3}	0.7	115

B. The correlation between each tuned hyperparameter and validation loss.