Movie Recommender System - Project on Information Retrieval and Web Search

Tanya Agarwal, and Shubham Sarda
Indian Institute of Technology, Delhi
(tt1180963, tt1180958)@iitd.ac.in

Abstract

This project implements a movie recommendation system for users in the MovieLens database. Our recommendation system extracts user data and movie genres from the MovieLens datasets, and uses IMDb descriptions of movies for improved analysis, extracted via Web Scraping. We provide recommendations based on the user's overall past preferences, based on their most recently watched films, as well as some basic recommendations in case of a new user who hasn't previously used Movie Lens. We have employed item-item collaborative filtering, user-user collaborative filtering, and multiple separate neural architectures that implement neural collaborative filtering. Further, we have performed 10-fold cross-validation on each of our architectures in order to evaluate their performance.

1. Introduction

Recommender systems may be considered a subset of information retrieval systems, based off predicting what a user will like and accordingly retrieving relevant items from the database for the user to peruse. Our movie recommender system applies this idea to a database of movies, and recommends the movies that it predicts the user is most likely to like or give a high rating to.

In our recommendation system, we provide three options to the user: (i) Recommendations based on past user preferences (ii) Recommendations similar to the most recently watched movie by the user, and (iii) Providing some suggestions to new users in the case of a cold-start problem, either based on some preferred genres/ preferred movie name inputted by the new user, or based on ratings given by existing users in the database.

In order to tailor our recommendations to a certain user - specifically to users existing in the Movie Lens database - we have observed their past user ratings and preferences given to movies that they have watched in the past. Accordingly, we suggest movies that have similar themes/ genres and content, or movies that other users similar to the target user have liked in the past. Also, we have created different neural architectures that utilize user and movie data in order to predict relevant movies that the user will like.

2. Related Work

2.1. Web Scraping

Web scraping [6] refers to a process of extracting data from the Web and often storing it for further use in a database, typically in a manner that it can be done automatically, that is, without the need of a human having to browse through links and copy relevant data manually. Web scraping can often be done by methods like HTML parsing [7], which allows for using the template of similar web-pages in extracting relevant data from the relevant section with ease and utilizing it for applications like database augmentation and expansion.

2.2. Item-Item Collaborative Filtering Systems

The collaborative filtering technique [2] is used to identify items best suited to a user by making predictions based off a large amount of data of ('collaboration' of) past user preferences. Item-item collaborative systems [1] aim to do this by finding the similarity between different items that exist in the database, and accordingly identifying similar ones to those that the user has preferred in the past. The rating given by user U to item I_i can be predicted as [3]:

$$rating(U, I_i) = \frac{\sum_j rating(U, I_j) * s_{ij}}{\sum_j s_{ij}}$$

2.3. User-User Collaborative Filtering Systems

Collaborative filtering as defined above makes recommendations based off a 'collaboration' or congregation of past user preferences. In the case of user-user collaborative filtering [2], the preference or rating that a user would give to an item is predicted by looking at the ratings given by some similar users to the same item.

Here, 'similarity' between two users a and b can be calculated by using the Pearson correlation coefficient [4] as :

$$Sim(a,b) = \frac{\sum_p (r_{ap} - \bar{r}_a)(r_{ab} - \bar{r}_b)}{\sqrt{\sum_p (r_{ap} - \bar{r}_a)^2}\sqrt{\sum_p (r_{bp} - \bar{r}_b)^2}}$$

where, p is an item available in the dataset that has been rated in the past by users a and b both.

The rating given by user u to item p can then be estimated [5] as:

$$r_{up} = \bar{r}_u + \frac{\sum_{i \in users} sim(u,i) * r_{ip}}{\sum_{i \in users} |sim(u,i)|}$$

where, r_{up} is the rating given by user u to item p, and $\bar{r}_u$ is the average rating given by user u to the movies they have watched in the past. Also, the $users$ set that user i belongs to is a specific number of the most-similar users to u.

2.4. Neural Collaborative Filtering Systems

Deep Learning has taken the world by storm. Since its emergence, its usage and popularity has been growing at an exponential rate. Most of the state-of-the-art results, be it computer vision or natural language processing subtasks, have been achieved by deep learning models. Taking inspiration from the Neural Collaborative Filtering model proposed by Xiangnan He et al.[10], we have evaluated several self implemented neural architectures. The conventional neural model used for collaborative filtering is given in Figure 1.

3. Our Dataset, Augmented Through Web Scraping

We have used the MovieLens 1M data set [8], which contains 1,000,000 ratings on 4,000 movies from 6,000 users, for user-user collaborative filtering and neural collaborative filtering.

The ratings section of the 1M dataset contains the following features:

1. movieId
2. userId
3. rating (5-star scale)
4. title
5. timestamp
6. genres

We have preferred the MovieLens dataset in place of datasets available from websites like IMDb since it provides individual user ratings for each movie on the site, and not just an average single rating like IMDb. Also, while this dataset does not contain very recent movies, we have still used it since our models are generalized and can be extended to a more recent dataset with ease.

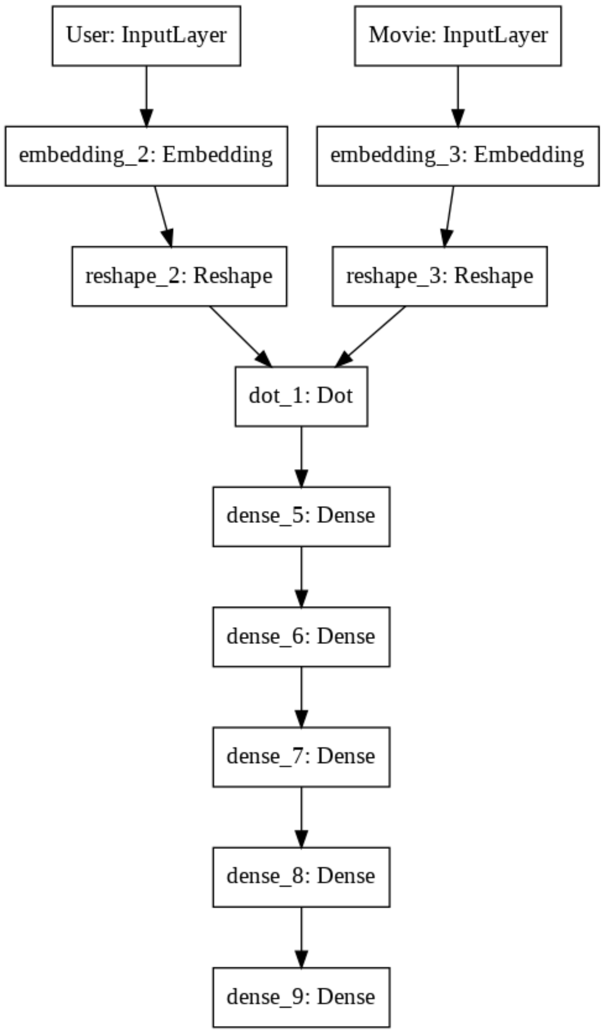
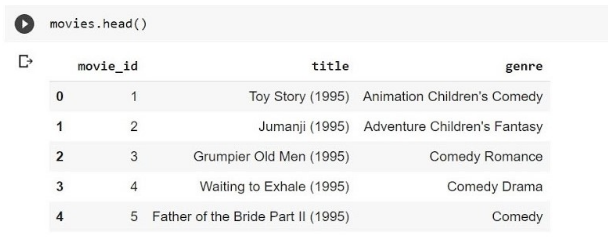


Figure 1. The conventional neural model used for collaborative filtering

In order to make our dataset for models like item-item collaborative filtering more comprehensive, we have performed web scraping from IMDb over all the movies of the MovieLens 1M dataset. We have extracted the movie description from the first result displayed in IMDb search for each movie in the MovieLens dataset. We have not used additional search results since IMDb only contains one unique correct result corresponding to a movie title, and using lower ranked search results had a higher chance of appending incorrect movie descriptions to our dataset. We have augmented the MovieLens dataset with these descriptions obtained via web scraping.

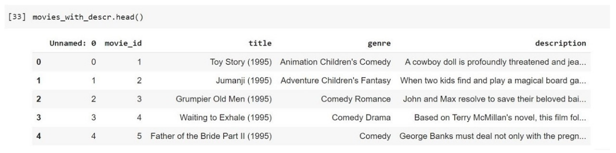
Figures 2 and 3 show our dataset before and after



```
movies.head()
```

	movie_id	title	genre
0	1	Toy Story (1995)	Animation Children's Comedy
1	2	Jumanji (1995)	Adventure Children's Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama
4	5	Father of the Bride Part II (1995)	Comedy

Figure 2. Our dataset before web scraping



```
[33] movies_with_descr.head()
```

	Unnamed: 0	movie_id	title	genre	description
0	0	1	Toy Story (1995)	Animation Children's Comedy	A cowboy doll is profoundly threatened and jea...
1	1	2	Jumanji (1995)	Adventure Children's Fantasy	When two kids find and play a magical board ga...
2	2	3	Grumpier Old Men (1995)	Comedy Romance	John and Max resolve to save their beloved bai...
3	3	4	Waiting to Exhale (1995)	Comedy Drama	Based on Terry McMillan's novel, this film fol...
4	4	5	Father of the Bride Part II (1995)	Comedy	George Banks must deal not only with the pregn...

Figure 3. Our dataset after web scraping

augmentation respectively.

4. Item-Item Collaborative Filtering Model

4.1. Our Model

We concatenate the web scraped description, movie title and genre to create the document embedding using the pretrained RoBERTa model. We've utilised the SentenceTransformer [18] library for the same. These models are quite suitable for tasks since they've been trained on the Siamese neural network for predicting semantic textual similarity between two sentences.

We hence define our function to calculate similarity between the user's preferred movies and other unwatched movies in the dataset in order to estimate ratings and recommend movies to the user.

Here we've created two functions, one for recommendation and other for evaluation on test sets.

1. Function - recommend($user_id, num_movies$): This function takes user˙id (unique identification tag for every user) and num_movies (number of movies to be recommended). As the name suggests, the function recommends 'num_movies' movies to the user with given user-id based on their past choices. We iterate over the whole movie dataset and rate each movie using the given formula. We've used cosine similarity to find the similarity between two movie embeddings. Thus we've utilised the ratings of watched movies and their similarity with the movie we desire to rate. While estimating the rating we only use top k most similar movies where k is an optimizable hyperparameter. We then sort the newly rated movies in decreasing order to recommend the top movies. In this function we iterate over all movies to predict their ratings. We also create a dictionary for structure with key as user id and values as the list of movies and ratings given by them. This decreases the computational cost. For every movie, we use the previous ratings of the user to predict the new rating . So the big O is $O(len(ratings) + len(movies) * Uavg)$, where $len(movies)$=total number of movies, $len(ratings)$=total number of ratings in the dataset, $Uavg$= average number of movies rated by each user.

2. Function - predict(test,train,k=10): This function takes the test dataset as input which only contains the user ids and movie ids. The train dataset contains user ids, movie ids and the corresponding ratings. We use the algorithm used in the above method to rate the unrated movies. However here we don't iterate over the whole movie dataset as we're not recommending and just filling out the entries in the test set. Here also we pass the train set in a dictionary structure to reduce computational cost. Then we iterate over the whole test set so for finding big O we'll just replace $len(movies)$ by $len(testset)$ in the previous expression. Thus the expression for big O is $O(len(ratings) + len(testset) * Uavg)$, where $len(movies)$=total number of movies, $len(ratings)$=total number of ratings in the dataset, $Uavg$= average number of movies rated by each user.

5. User-User Collaborative Filtering Model

5.1. Our Model

We have calculated the similarity between two users using the Pearson correlation coefficient - i.e. what may be considered cosine similarity for centered vectors. [9] For a movie the user hasn't watched, a weighted average of the ratings given by top-k most similar other users to that movie is taken, where 'k' is an optimizable hyperparameter. We have used the formula described in section 2.3 in order to calculate this using our dataset, and have estimated user rating by considering the top $k = 100$ users that are similar to u.

We have then recommended top $num - movies$ movies to user u present in the MovieLens database, by sorting the predicted ratings for unwatched movies in descending order, as our recommendations to the user based on this model.

After creating a dictionary containing $(user, movieId) : (rating)$ values in $O(len(rating))$ time, calculating the similarity between 2 users takes $O(len(movies))$ time, where $len(rating)$ is the number of ratings available in the dataset, and $len(movies)$ is the number of movies in the MovieLens 1M dataset.

Predicting user ratings for a single movie after a one-time dictionary creation as described above hence takes $O(len(movies)) * O(len(ratings)) + O(k)$ time, where k is the number of most-similar users in the dataset that are considered in predicting rating. The $O(len(movies)) * O(len(ratings))$ time helps in creation of a similarity dictionary between user us and every other user, after which a linear traverse over the top-k similar users and $O(1)$ rating extraction time from the predefined dictionary gives this time complexity.

The overall recommender system hence takes $O(len(movies))$ time to find the unwatched movies by user u, and $[O(len(movies)) * O(len(ratings)) + O(k)] * O(unwatchedMovies)$ time overall for recommending new movies to the user.

Due to the higher time complexity of this recommendation function with very large number of movies, we have only used a small test set of 10k ratings instead of the 100k ratings set used for other model evaluations.

6. Neural Collaborative Filtering Models

6.1. Our Models

We have used Neural Collaborative filtering as described in section 2.4. We have built 8 neural architectures for our project. We have described each of these in the following points:

1. Neural Collaborative Filtering - Model 1 (NCF1): This model is pretty much similar to the basic neural collaborative framework used in [11]. We have kept the number of hidden layers and hidden units as the default size mentioned on their GitHub repository webpage. We have initially created 2 embedding matrices, one for users and other for movies. These are then merged using dot product followed by several fully connected layers. The final layer of a single hidden unit gives the predicted rating. Although much of the architecture is similar but we have incorporated a few changes like modification of number of factors and output layer activation function. They had used ReLU [12] function for all hidden layers and sigmoid for output layer followed by scaling. We have chosen to use ReLU even for the output layer as sigmoid produces clustered towards 0 or 5 due to exponential nature. Figure 2 represents the neural architecture that we have used here.

2. Neural Collaborative Filtering - Model 2 (NCF2): We have added several modifications to the base model. We have added Batch Normalization [13] after every fully connected layer for better scaling and results. This allows us to train deeper networks with lesser risks of vanishing as well as exploding gradient problem. Deeper networks perform better in learning the user and movie latent factors/features. We have also added Dropout [14] layers in between to avoid overfitting on the training set. In the merging step, conventionally dot product has been used in the previous implementations. We have experimented with addition, concatenation, element wise multiplication for proper merging of semantic features of movies and users. We finally used concatenation as it gave the best results. Further instead of just using dense layers after the merge step, we propose additional fully connected layers between the initial embeddings and the merging step for better learning. Another modification we have introduced is the weight initialization in the movie embedding layer. We have used a vectorized form of the movie descriptions to initialize the movie embedding matrix instead of random initialization. Sentence embeddings of 1024 dimensions were generated form the Sentence-Roberta model as done in Item-Item collaborative filtering. This head start resulted in improved final results as well as a faster convergence to the minima. Figure 3 represents the neural architecture that we have used here.

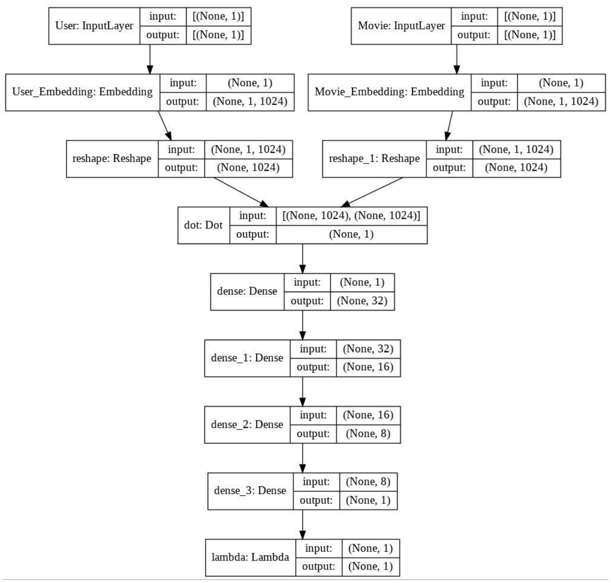


Figure 4. Neural Collaborative Filtering - Model 1

3. Neural Collaborative Filtering - Model 3 (NCF3): In this model, the initial architecture upto the concatenation is the same as the previous model. However after that instead of simply using a dense decoder network with [512, 256..8,1] hidden units, we've used a U-net[15] sort of architecture. Similar to the Unet, we first increase then decrease the number of hidden units. The corresponding hidden layers with the same hidden units are connected using skip-residual connections. Figure 4 represents the neural architecture we have used here. Figure 4 represents

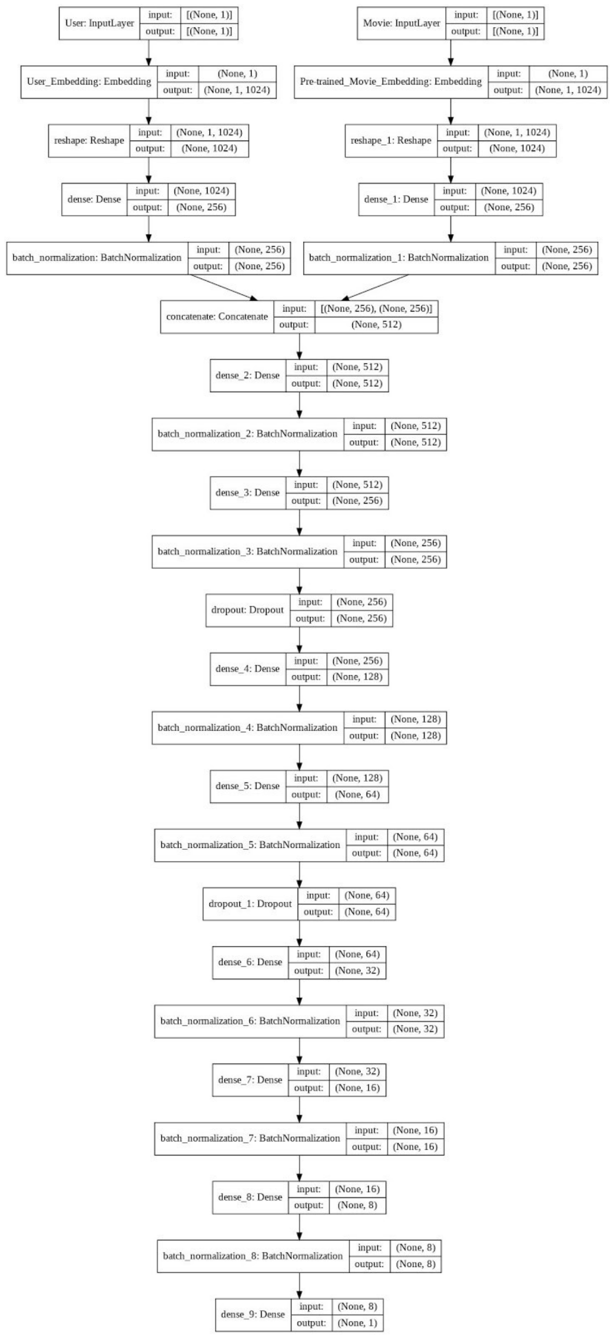


Figure 5. Neural Collaborative Filtering - Model 2

the neural architecture that we have used here.

4. Neural Collaborative Filtering - Model 4 (NCF4): Here we have tried leveraging word vectors based embeddings followed by a stacked Bidirectional LSTM [16]. The movie descriptions were cleaned through removal of punctuation, numbers and stop words followed by lowercasing. Following all the preprocessing, the maximum length of description was

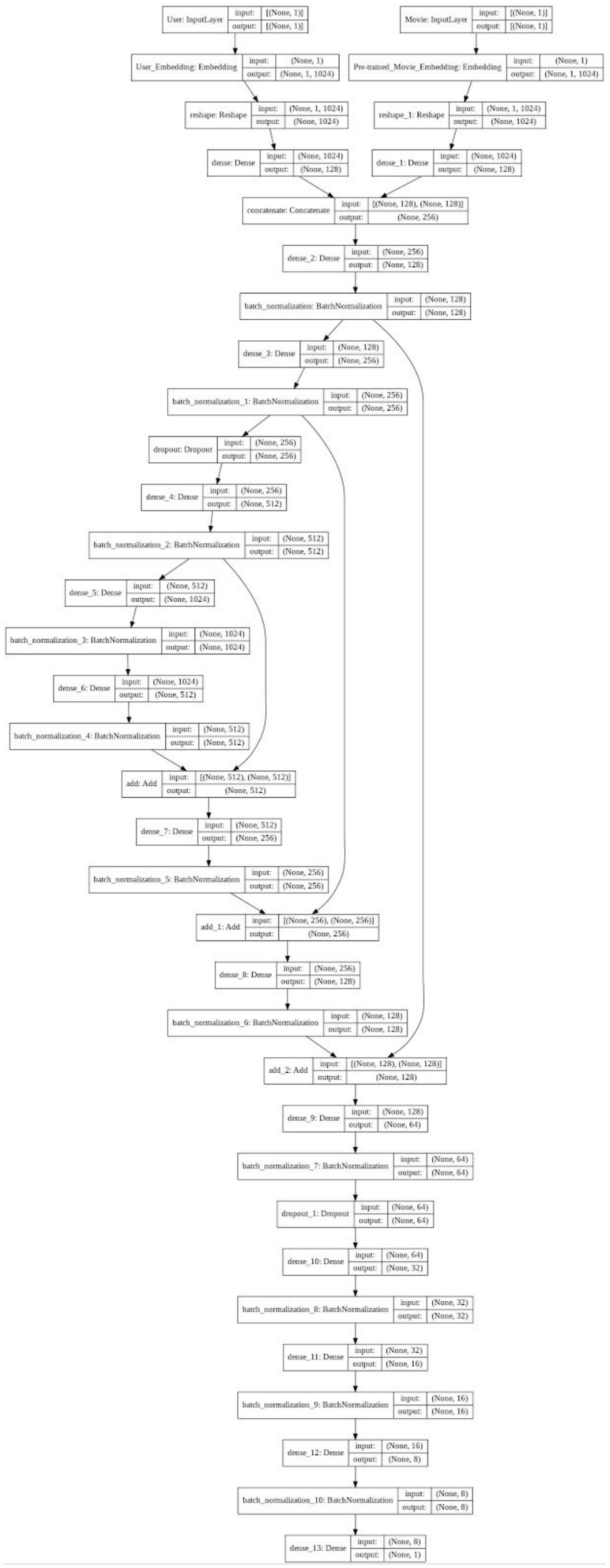


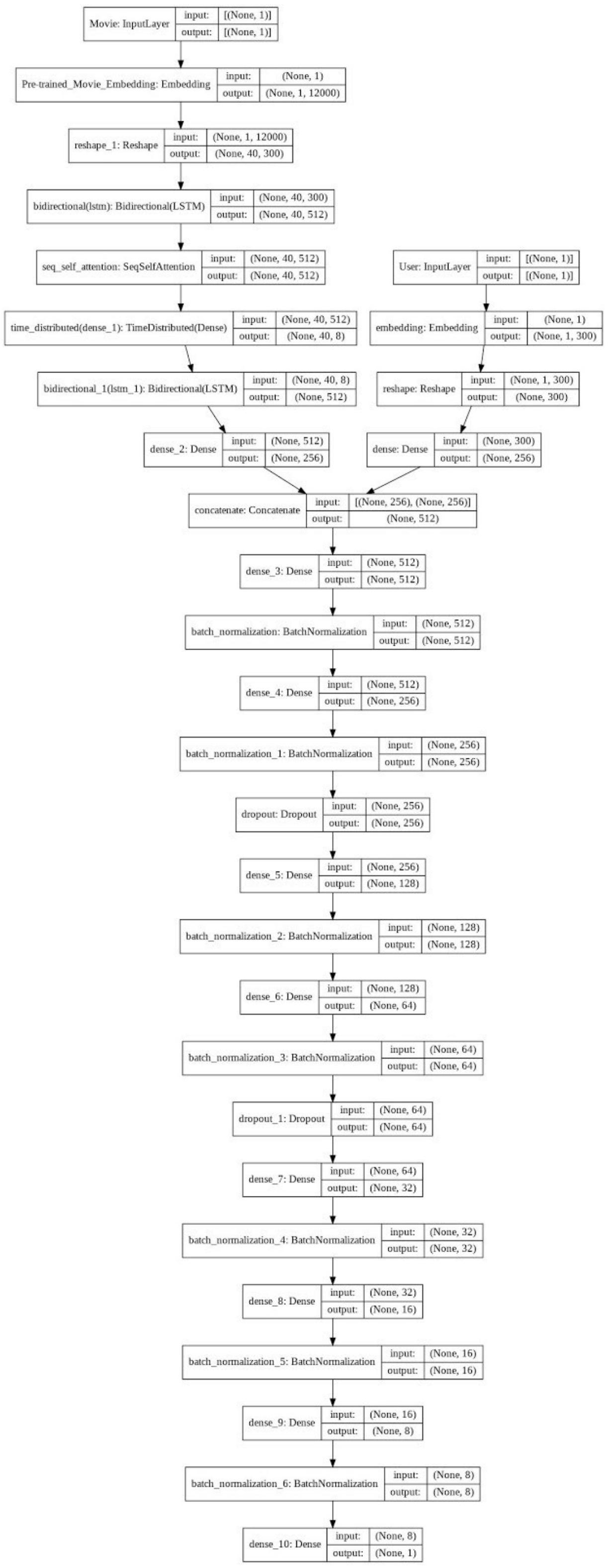
Figure 6. Neural Collaborative Filtering - Model 3

found to be 40. We had used spaCy word embeddings [17] of 300 dimensions. So 40 embeddings of 300 dimensions were stacked together and reshaped as [30*400] sized vector to fit the movie embedding layer. Cleaned movies descriptions with lengths less than 40 were stacked with zero-padded vectors to fit the embedding layer. The embedding layer was then reshaped and fed into a 40 cell bidirectional LSTM layer. For the first LSTM layer, we've returned all the 40 sequences to allow another LSTM layer Attention mechanism was employed to improve the learning capability of the LSTM encoder network by focussing on more relevant/important terms. This was followed by a time distributed dense layer and another bidirectional LSTM layer. For this LSTM layer, only the final sequence was returned which was then fed into a dense layer. This was followed by the concatenation of the user and movie vectors. After concatenation, decoding architecture is the same as that used in the model 2. Figure 5 represents the neural architecture that we have used here.

5. Neural Collaborative Filtering - Model 5 (NCF5): In this architecture also, we have used Bidirectional LSTM layers with attention mechanism. Thus the initial encoding architecture upto concatenation is the same as that used in the previous model. Following concatenation, we use a Unet style network with skip connections as done in model 3. Figure 6 represents the neural architecture that we have used here.

6. Neural Collaborative Filtering - Model 6 (NCF6): In the model, we have tried combining LSTMs with the transformer based movie description embeddings. Here we feed the Sentence Roberta embeddings to the Bidirectional LSTM layer. Since we have a collective embedding for the whole description, this is a one-to-many type of LSTM encoder with a single time step in the input. This followed by an attention layer and a second LSTM layer. The output is fed into a dense layer which is concatenated with the user vectors. We then follow a similar decoding architecture used in model 2 and 4 to reach the final output layer with a single unit which estimates the rating. Figure 7 represents the neural architecture that we have used here.

7. Neural Collaborative Filtering - Model 7 (NCF7): In this model also we have tried to use Bidirectional LSTMs with the roverta embeddings. The initial LSTM encoding network is same as the one used in the previous model. After concatenation, we have adopted a Unet style network with skip connection between the corresponding layers with the same hidden units.


Movie: InputLayer | input: [(None, 1)] | output: [(None, 1)]
Pre-trained_Movie_Embedding: Embedding | input: (None, 1) | output: (None, 1, 12000)
reshape_1: Reshape | input: (None, 1, 12000) | output: (None, 40, 300)
bidirectional(lstm): Bidirectional(LSTM) | input: (None, 40, 300) | output: (None, 40, 512)
seq_self_attention: SeqSelfAttention | input: (None, 40, 512) | output: (None, 40, 512)
time_distributed(dense_1): TimeDistributed(Dense) | input: (None, 40, 512) | output: (None, 40, 8)
bidirectional_1(lstm_1): Bidirectional(LSTM) | input: (None, 40, 8) | output: (None, 512)
dense_2: Dense | input: (None, 512) | output: (None, 256)
User: InputLayer | input: [(None, 1)] | output: [(None, 1)]
embedding: Embedding | input: (None, 1) | output: (None, 1, 300)
reshape: Reshape | input: (None, 1, 300) | output: (None, 300)
dense: Dense | input: (None, 300) | output: (None, 256)
concatenate: Concatenate | input: [(None, 256), (None, 256)] | output: (None, 512)
dense_3: Dense | input: (None, 512) | output: (None, 512)
batch_normalization: BatchNormalization | input: (None, 512) | output: (None, 512)
dense_4: Dense | input: (None, 512) | output: (None, 256)
batch_normalization_1: BatchNormalization | input: (None, 256) | output: (None, 256)
dropout: Dropout | input: (None, 256) | output: (None, 256)
dense_5: Dense | input: (None, 256) | output: (None, 128)
batch_normalization_2: BatchNormalization | input: (None, 128) | output: (None, 128)
dense_6: Dense | input: (None, 128) | output: (None, 64)
batch_normalization_3: BatchNormalization | input: (None, 64) | output: (None, 64)
dropout_1: Dropout | input: (None, 64) | output: (None, 64)
dense_7: Dense | input: (None, 64) | output: (None, 32)
batch_normalization_4: BatchNormalization | input: (None, 32) | output: (None, 32)
dense_8: Dense | input: (None, 32) | output: (None, 16)
batch_normalization_5: BatchNormalization | input: (None, 16) | output: (None, 16)
dense_9: Dense | input: (None, 16) | output: (None, 8)
batch_normalization_6: BatchNormalization | input: (None, 8) | output: (None, 8)
dense_10: Dense | input: (None, 8) | output: (None, 1)
Figure 7. Neural Collaborative Filtering - Model 4

The is followed by the final output layer predicting the movie rating. Figure 8 represents the neural architecture that we have used here.

7. Similar Movies to the Most-Recently-Watched Movie

Sometimes, the user may have just completed watching a movie and might hence want to watch similar movies to the movie they have just watched. In this case, we have used our item-item collaborative filtering function to return the top $num - movies$ movies which are most-similar to the movie of the user having the most recent timestamp, and recommended the same to the user. This idea stems from the Netflix recommendation system of ”If you liked this movie, you may also like” each time a user finishes watching a movie.

8. Recommendations in Cold Start Case

In the case where a new user has joined the database, their previous history will not be available in the MovieLens dataset. Hence in this case, it will not be possible to predict user preferences by the models that we have described previously (that is, there is a ”Cold Start” problem in this case which we need to handle). To deal with such a case, we have provided 2 options:

1. The overall top-rated $num - movies$ movies will be returned to the user. To implement this, we have calculate the average rating as well as average timestamp of movies in our database, and used that to sort the movies first by highest rating, and then by most-recent timestamp.

2. The user is asked to enter their favourite movie’s name. Using that, we return the top $num - movies$ movies which are similar to this movie, as found by our item-item collaborative filtering model.

9. Results

Model	RMSE	—
	Mean	SD
User-User	1.118	0.019
Item-Item	1.59	0.0288
NCF1	1.11	.003
NCF2	0.885	0.005
NCF3	0.8733	0.025
NCF4	0.8731	0.0128
NCF5	0.8736	0.008
NCF6	0.8672	0.006
NCF7	**0.866**	0.024

Model	NDCG					
Model	3		4		5	
	Mean	SD	Mean	SD	Mean	SD
User-User	0.966	0.0032	0.926	0.0048	0.8243	0.012
Item-Item	0.9186	.0009	0.8451	0.0013	0.672	0.0019
NCF1	0.915	0.009	0.841	0.015	0.673	0.026
NCF2	0.956	0.0004	0.912	0.001	0.79	0.001
NCF3	0.9571	0.001	0.9155	0.001	0.797	0.003
NCF4	0.9579	0.001	0.9159	0.001	0.7983	0.004
NCF5	0.9576	0.0004	0.9151	0.0009	0.7969	0.0007
NCF6	**0.9593**	0.005	**0.9173**	0.0005	**0.8009**	0.0016
NCF7	0.9580	0.0012	0.9165	0.0015	0.7997	0.0027

Model	MAP					
Model	3		4		5	
	Mean	SD	Mean	SD	Mean	SD
User-User	0.964	0.005	0.863	0.0087	0.706	0.016
Item-Item	0.912	0.0015	0.749	0.002	0.49	0.004
NCF1	0.907	0.009	0.744	0.02	0.497	0.03
NCF2	0.955	0.0005	0.8486	0.002	0.655	0.002
NCF3	0.957	0.001	0.0853	0.002	0.6657	0.003
NCF4	0.057	0.001	0.08541	0.00e1	0.6664	0.006
NCF5	0.95690	0.0008	0.08531	0.002	**0.671**	0.011
NCF6	**0.9588**	0.0005	0.8564	0.0017	0.6698	0.0025
NCF7	0.9577	0.0020	**0.8851**	0.0025	0.6681	0.0041

We have performed 10-fold cross validation on the 1M dataset using our models, wherein we evaluated the models 10 times by dividing the dataset into train and test sets in a 90:10 ratio each time. The final results of our evaluations have been summarized in the above tables.

We’ve evaluated using NDCG(Normalized Discounted Cumulative Gain), RMSE(root mean squared error) and MAP(Mean Average Precision) metrics. Several more results are available on the link at [19]. We’ve created Qrel files by sorting the test set by ground truth ratings. We’ve experimented with three threshold ratings- 3, 4 and 5 to consider a movie relevant. For creating the results file, we sorted the test set on the basis of predicted ratings. Then we used the trec evaluation [22] software for finding these metrics.

In terms of RMSE, current sota model for this dataset is the Bayesian Time Flipped SVD [20] with rmse score of 8.18. In terms of NDCG, current sota model is KTUP model [21] with a NDCG score of 0.699. So on comparing, our rmse results for several neural architecture are very close to the latest sota results, For NDCG, although they haven't specified the relevance threshold used for NDCG calculation, our neural collaborative filtering models are in line with the current sota results. (Due to small test evaluation for user-user filtering, we've not used its results for comparison)

10. Statistical Tests and Analysis of Results

We've used Paired Student's t-test and Wilcoxon Signed Rank Test as the statistical significance test. We've performed the test for all the possible combinations of our models. We've omitted user user collaborative filtering as it was run on a small test set. The test statistic is mentioned as stat and the p value is mentioned as p. We have used the significance level as 0.01%. The Paired Student's t-test and Wilcoxon Signed-Rank Test results are as follows:

1. Item-Item and NCF1 Paired Student's t-test: stat=-997.089, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2393057.000, p=0.000 Probably different distributions

2.Item-Item and NCF2 Paired Student's t-test: stat=-801.802, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=8878679.000, p=0.000 Probably different distributions

3. Item-Item and NCF3 Paired Student's t-test: stat=-828.358, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test:stat=2525690.000, p=0.000 Probably different distributions

4. Item-Item and NCF4 Paired Student's t-test: stat=-809.589, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=3176116.000, p=0.000 Probably different distributions

5. Item-Item and NCF5: Paired Student's t-test: stat=-823.040, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2812285.000, p=0.000 Probably different distributions

6. Item-Item and NCF6 Paired Student's t-test: stat=-793.528, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=5068971.000, p=0.000 Probably different distributions

7. Item-Item and NCF 7 Paired Student's t-test: stat=-797.185, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=3648011.000, p=0.000 Probably different distributions

8. NCF1 and NCF2 Paired Student's t-test: stat=-1.697, p=0.090 Probably the same distributions
Wilcoxon Signed Rank Test: tat=2351941548.000, p=0.000 Probably different distributions

9. NCF1 and NCF3 Paired Student's t-test: stat=2351941548.000, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2380378226.000, p=0.000 Probably different distributions

10. NCF1 and NCF4 Paired Student's t-test: stat=0.013, p=0.990 Probably the same distributions
Wilcoxon Signed Rank Test: tat=2366425636.500, p=0.000 Probably different distributions

11. NCF1 and NCF5 Paired Student's t-test: stat=-7.192, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2287994440.000, p=0.000 Probably different distribution

12. NCF1 and NCF6 Paired Student's t-test: stat=-15.464, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2205050138.500, p=0.000 Probably different distributions

13. NCF1 and NCF7 Paired Student's t-test: stat=-2.919, p=0.004 Probably different distributions
Wilcoxon Signed Rank Test: stat=2341077828.000, p=0.000 Probably different distributions

14. NCF2 and NCF3 Paired Student's t-test: stat=3.408, p=0.001 Probably different distributions
Wilcoxon Signed Rank Test: stat=2423307318.500, p=0.000 Probably different distributions

15. NCF2 and NCF4 Paired Student's t-test: stat=3.544, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2442572049.500, p=0.000 Probably different distributions

16. NCF2 and NCF5 Paired Student's t-test: stat=-11.222, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2432135675.000, p=0.000 Probably different distributions

17. NCF2 and NCF6 Paired Student's t-test: stat=-28.617, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2189101868.000, p=0.000 Probably different distributions

18. NCF2 and NCF7 Paired Student's t-test: stat=-2.649, p=0.008 Probably different distributions
Wilcoxon Signed Rank Test: stat=2484413369.000, p=0.068 Probably the same distribution

19. NCF3 and NCF4 Paired Student's t-test: stat=0.616, p=0.538 Probably the same distributions
Wilcoxon Signed Rank Test: stat=2483053036.500, p=0.049 Probably the same distribution

20. NCF3 and NCF5 Paired Student's t-test: stat=-18.609, p=0.000 Probably different distributions

Wilcoxon Signed Rank Test: stat=2316447093.500, p=0.000 Probably different distributions

21. NCF3 and NCF6 Paired Student's t-test: stat=-40.024, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2036533250.500, p=0.000 Probably different distributions

22. NCF3 and NCF7 Paired Student's t-test: stat=-7.374, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2393852634.000, p=0.000 Probably different distributions

23. NCF4 and NCF5 Paired Student's t-test: stat=-24.901, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2259637424.000, p=0.000 Probably different distributions

25. NCF4 and NCF6 Paired Student's t-test: stat=-47.513, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=1941500624.500, p=0.000 Probably different distributions

26. NCF4 and NCF7 Paired Student's t-test: stat=-8.618, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2389990058.000, p=0.000 Probably different distributions

27. NCF5 and NCF6 Paired Student's t-test: stat=-25.545, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2154575601.500, p=0.000 Probably different distributions

28. NCF5 and NCF7 Paired Student's t-test: stat=11.633, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2421816992.000, p=0.000 Probably different distributions

29. NCF6 and NCF7 Paired Student's t-test: stat=37.594, p=0.000 Probably different distributions
Wilcoxon Signed Rank Test: stat=2085442159.000, p=0.000 Probably different distributions

11. Acknowledgements

We would like to thank our professor, Srikanta Bedathur sir, our instructor for our Information Retrieval course, for giving us the opportunity to work on this topic and being available for help and guidance at every step of the process. We would also like to thank our TAs for the course, Vinayak Gupta and Mohit Gupta, for always being available to clarify our doubts.

References

[1] Item-item collaborative filtering article from https://en.wikipedia.org/wiki/Item-item˙collaborative˙filtering.

[2] Collaborative filtering Wikipedia from https://en.wikipedia.org/wiki/Collaborative˙filtering/.

[3] Formula for item item collaborative filtering: https://www.geeksforgeeks.org/item-to-item-based-collaborative-filtering/

[4] Pearson correlation coefficient from https://en.wikipedia.org/wiki/Pearson˙correlation˙coefficient

[5] https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26

[6] What is web scraping, from https://www.scrapinghub.com/

[7] https://en.wikipedia.org/wiki/Web˙scraping

[8] https://grouplens.org/datasets/movielens/1m/

[9] https://stats.stackexchange.com/questions/235673/is-there-any-relationship-among-cosine-similarity-pearson-correlation-and-z-sc

[10] https://arxiv.org/abs/1708.05031

[11] https://arxiv.org/pdf/1708.05031.pdf

[12] ReLU function, https://www.cs.toronto.edu/ fritz/absps/reluICML.pdf

[13] https://arxiv.org/abs/1502.03167

[14] https://dl.acm.org/doi/10.5555/2627435.2670313

[15] https://arxiv.org/abs/1505.04597

[16] https://www.bioinf.jku.at/publications/older/2604.pdf

[17] Industrial strength NLP https://spacy.io/

[18] https://arxiv.org/pdf/1908.10084v1.pdf]

[19] Our remaining results are at https://docs.google.com/spreadsheets/d/1KVXD6MGQhXYBByOostV
9GHZVUo3o/edit?usp=sharing

[20] https://arxiv.org/pdf/1905.01395v1.pdf

[21] https://arxiv.org/pdf/1902.06236v1.pdf

[22] https://github.com/usnistgov/trec˙eval

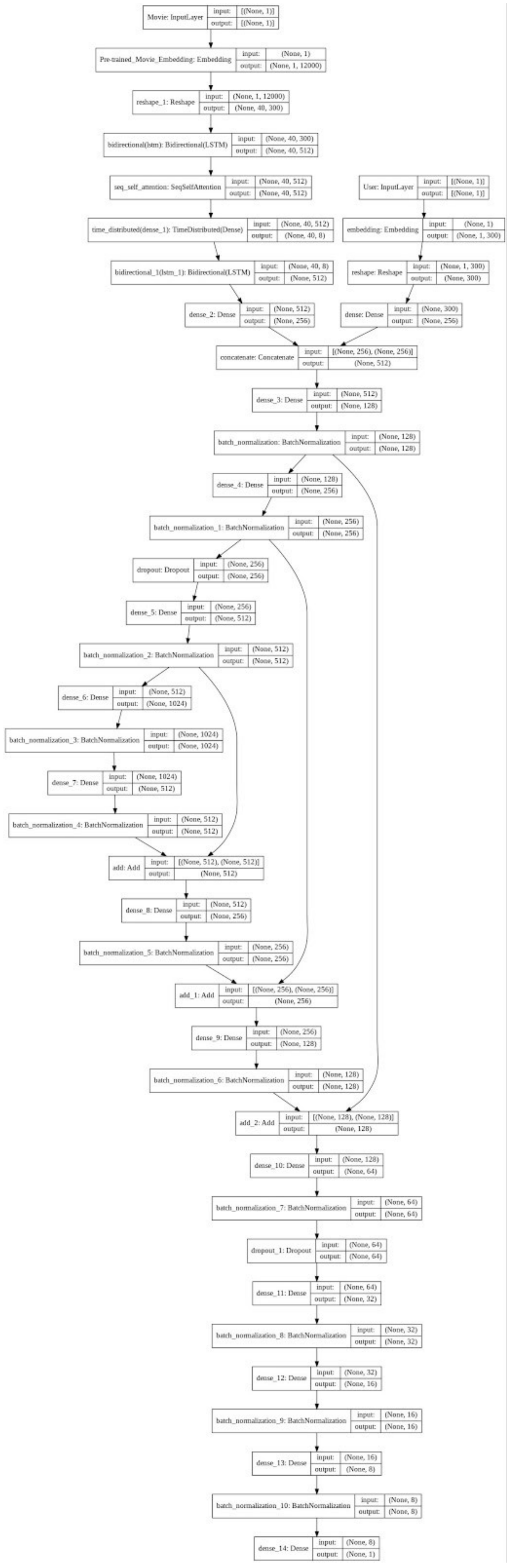

Movie: InputLayer input: [(None, 1)] output: [(None, 1)]
Pre-trained_Movie_Embedding: Embedding input: (None, 1) output: (None, 1, 12000)
reshape_1: Reshape input: (None, 1, 12000) output: (None, 40, 300)
bidirectional(lstm): Bidirectional(LSTM) input: (None, 40, 300) output: (None, 40, 512)
seq_self_attention: SeqSelfAttention input: (None, 40, 512) output: (None, 40, 512)
User: InputLayer input: [(None, 1)] output: [(None, 1)]
time_distributed(dense_1): TimeDistributed(Dense) input: (None, 40, 512) output: (None, 40, 8)
embedding: Embedding input: (None, 1) output: (None, 1, 300)
bidirectional_1(lstm_1): Bidirectional(LSTM) input: (None, 40, 8) output: (None, 512)
reshape: Reshape input: (None, 1, 300) output: (None, 300)
dense_2: Dense input: (None, 512) output: (None, 256)
dense: Dense input: (None, 300) output: (None, 256)
concatenate: Concatenate input: [(None, 256), (None, 256)] output: (None, 512)
dense_3: Dense input: (None, 512) output: (None, 128)
batch_normalization: BatchNormalization input: (None, 128) output: (None, 128)
dense_4: Dense input: (None, 128) output: (None, 256)
batch_normalization_1: BatchNormalization input: (None, 256) output: (None, 256)
dropout: Dropout input: (None, 256) output: (None, 256)
dense_5: Dense input: (None, 256) output: (None, 512)
batch_normalization_2: BatchNormalization input: (None, 512) output: (None, 512)
dense_6: Dense input: (None, 512) output: (None, 1024)
batch_normalization_3: BatchNormalization input: (None, 1024) output: (None, 1024)
dense_7: Dense input: (None, 1024) output: (None, 512)
batch_normalization_4: BatchNormalization input: (None, 512) output: (None, 512)
add: Add input: [(None, 512), (None, 512)] output: (None, 512)
dense_8: Dense input: (None, 512) output: (None, 256)
batch_normalization_5: BatchNormalization input: (None, 256) output: (None, 256)
add_1: Add input: [(None, 256), (None, 256)] output: (None, 256)
dense_9: Dense input: (None, 256) output: (None, 128)
batch_normalization_6: BatchNormalization input: (None, 128) output: (None, 128)
add_2: Add input: [(None, 128), (None, 128)] output: (None, 128)
dense_10: Dense input: (None, 128) output: (None, 64)
batch_normalization_7: BatchNormalization input: (None, 64) output: (None, 64)
dropout_1: Dropout input: (None, 64) output: (None, 64)
dense_11: Dense input: (None, 64) output: (None, 32)
batch_normalization_8: BatchNormalization input: (None, 32) output: (None, 32)
dense_12: Dense input: (None, 32) output: (None, 16)
batch_normalization_9: BatchNormalization input: (None, 16) output: (None, 16)
dense_13: Dense input: (None, 16) output: (None, 8)
batch_normalization_10: BatchNormalization input: (None, 8) output: (None, 8)
dense_14: Dense input: (None, 8) output: (None, 1)

Figure 8. Neural Collaborative Filtering - Model 5

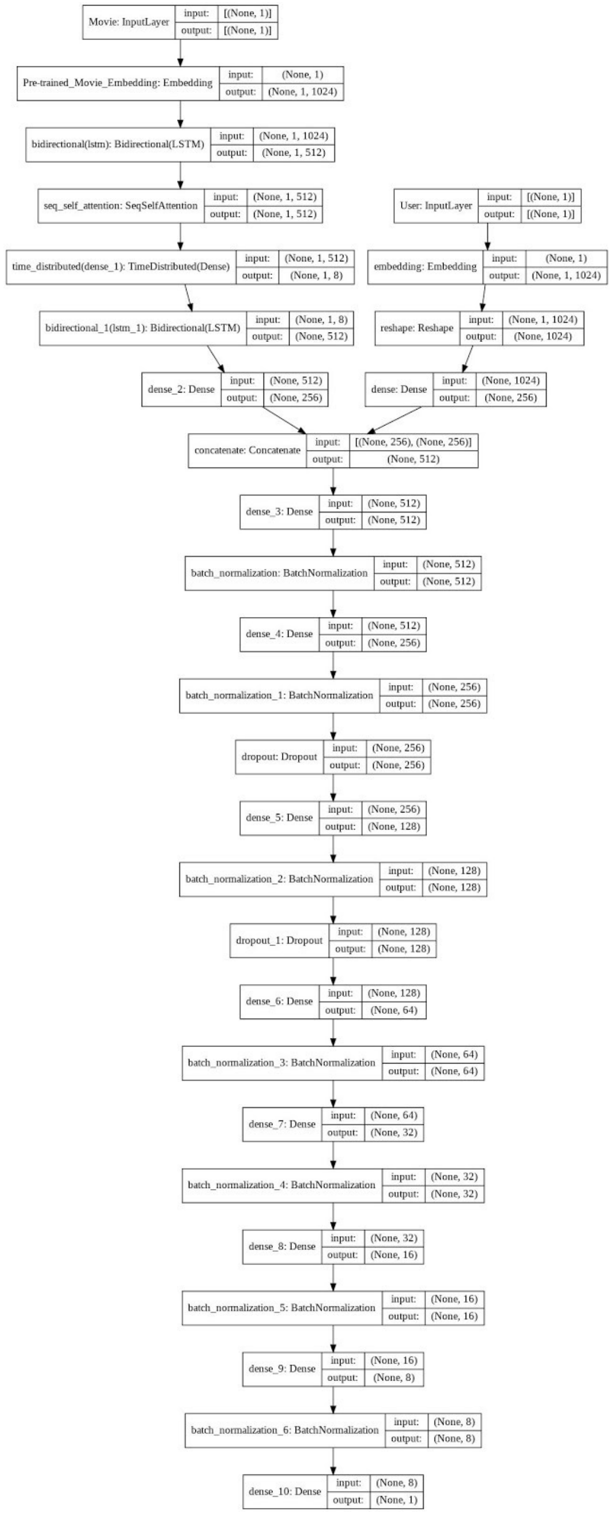

Movie: InputLayer input: [(None, 1)] output: [(None, 1)]
Pre-trained_Movie_Embedding: Embedding input: (None, 1) output: (None, 1, 1024)
bidirectional(lstm): Bidirectional(LSTM) input: (None, 1, 1024) output: (None, 1, 512)
seq_self_attention: SeqSelfAttention input: (None, 1, 512) output: (None, 1, 512)
User: InputLayer input: [(None, 1)] output: [(None, 1)]
time_distributed(dense_1): TimeDistributed(Dense) input: (None, 1, 512) output: (None, 1, 8)
embedding: Embedding input: (None, 1) output: (None, 1, 1024)
bidirectional_1(lstm_1): Bidirectional(LSTM) input: (None, 1, 8) output: (None, 512)
reshape: Reshape input: (None, 1, 1024) output: (None, 1024)
dense_2: Dense input: (None, 512) output: (None, 256)
dense: Dense input: (None, 1024) output: (None, 256)
concatenate: Concatenate input: [(None, 256), (None, 256)] output: (None, 512)
dense_3: Dense input: (None, 512) output: (None, 512)
batch_normalization: BatchNormalization input: (None, 512) output: (None, 512)
dense_4: Dense input: (None, 512) output: (None, 256)
batch_normalization_1: BatchNormalization input: (None, 256) output: (None, 256)
dropout: Dropout input: (None, 256) output: (None, 256)
dense_5: Dense input: (None, 256) output: (None, 128)
batch_normalization_2: BatchNormalization input: (None, 128) output: (None, 128)
dropout_1: Dropout input: (None, 128) output: (None, 128)
dense_6: Dense input: (None, 128) output: (None, 64)
batch_normalization_3: BatchNormalization input: (None, 64) output: (None, 64)
dense_7: Dense input: (None, 64) output: (None, 32)
batch_normalization_4: BatchNormalization input: (None, 32) output: (None, 32)
dense_8: Dense input: (None, 32) output: (None, 16)
batch_normalization_5: BatchNormalization input: (None, 16) output: (None, 16)
dense_9: Dense input: (None, 16) output: (None, 8)
batch_normalization_6: BatchNormalization input: (None, 8) output: (None, 8)
dense_10: Dense input: (None, 8) output: (None, 1)

Figure 9. Neural Collaborative Filtering - Model 6

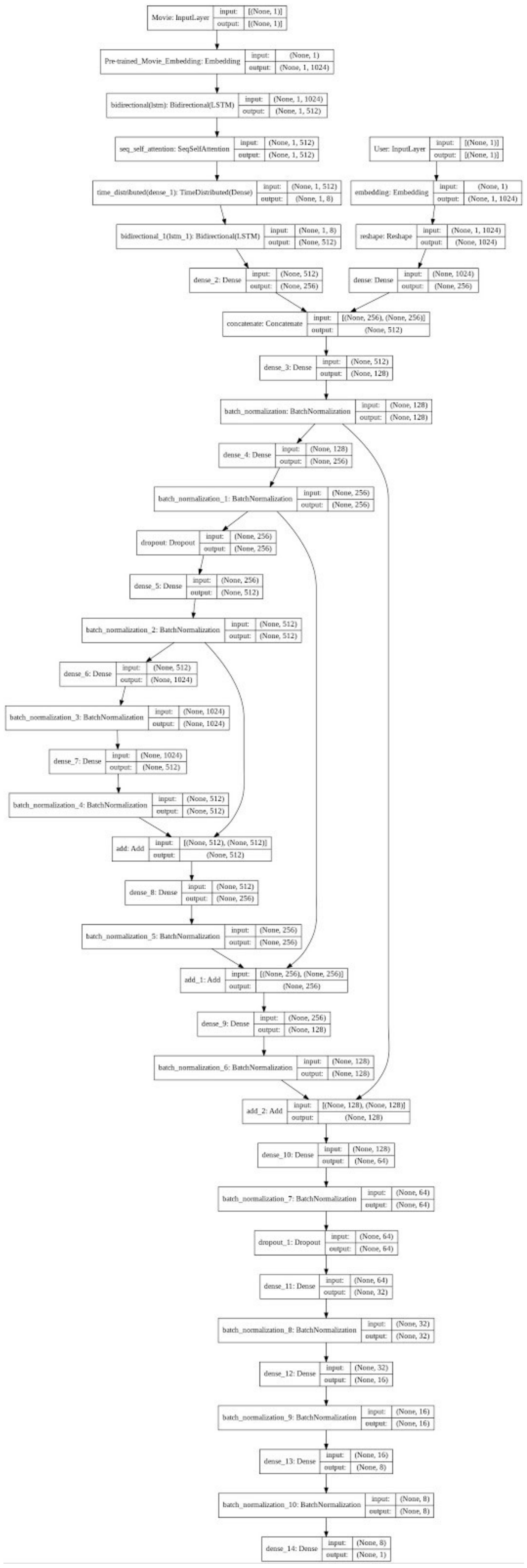

Movie: InputLayer | input: [(None, 1)] | output: [(None, 1)]
Pre-trained_Movie_Embedding: Embedding | input: (None, 1) | output: (None, 1, 1024)
bidirectional(lstm): Bidirectional(LSTM) | input: (None, 1, 1024) | output: (None, 1, 512)
seq_self_attention: SeqSelfAttention | input: (None, 1, 512) | output: (None, 1, 512)
time_distributed(dense_1): TimeDistributed(Dense) | input: (None, 1, 512) | output: (None, 1, 8)
bidirectional_1(lstm_1): Bidirectional(LSTM) | input: (None, 1, 8) | output: (None, 512)
dense_2: Dense | input: (None, 512) | output: (None, 256)
User: InputLayer | input: [(None, 1)] | output: [(None, 1)]
embedding: Embedding | input: (None, 1) | output: (None, 1, 1024)
reshape: Reshape | input: (None, 1, 1024) | output: (None, 1024)
dense: Dense | input: (None, 1024) | output: (None, 256)
concatenate: Concatenate | input: [(None, 256), (None, 256)] | output: (None, 512)
dense_3: Dense | input: (None, 512) | output: (None, 128)
batch_normalization: BatchNormalization | input: (None, 128) | output: (None, 128)
dense_4: Dense | input: (None, 128) | output: (None, 256)
batch_normalization_1: BatchNormalization | input: (None, 256) | output: (None, 256)
dropout: Dropout | input: (None, 256) | output: (None, 256)
dense_5: Dense | input: (None, 256) | output: (None, 512)
batch_normalization_2: BatchNormalization | input: (None, 512) | output: (None, 512)
dense_6: Dense | input: (None, 512) | output: (None, 1024)
batch_normalization_3: BatchNormalization | input: (None, 1024) | output: (None, 1024)
dense_7: Dense | input: (None, 1024) | output: (None, 512)
batch_normalization_4: BatchNormalization | input: (None, 512) | output: (None, 512)
add: Add | input: [(None, 512), (None, 512)] | output: (None, 512)
dense_8: Dense | input: (None, 512) | output: (None, 256)
batch_normalization_5: BatchNormalization | input: (None, 256) | output: (None, 256)
add_1: Add | input: [(None, 256), (None, 256)] | output: (None, 256)
dense_9: Dense | input: (None, 256) | output: (None, 128)
batch_normalization_6: BatchNormalization | input: (None, 128) | output: (None, 128)
add_2: Add | input: [(None, 128), (None, 128)] | output: (None, 128)
dense_10: Dense | input: (None, 128) | output: (None, 64)
batch_normalization_7: BatchNormalization | input: (None, 64) | output: (None, 64)
dropout_1: Dropout | input: (None, 64) | output: (None, 64)
dense_11: Dense | input: (None, 64) | output: (None, 32)
batch_normalization_8: BatchNormalization | input: (None, 32) | output: (None, 32)
dense_12: Dense | input: (None, 32) | output: (None, 16)
batch_normalization_9: BatchNormalization | input: (None, 16) | output: (None, 16)
dense_13: Dense | input: (None, 16) | output: (None, 8)
batch_normalization_10: BatchNormalization | input: (None, 8) | output: (None, 8)
dense_14: Dense | input: (None, 8) | output: (None, 1)

Figure 10. Neural Collaborative Filtering - Model 7