RNN Based Movie Recommender System for Groups

Ashish Kayastha[1,*]

[1]Central Michigan University, Department of Computer Science, Mt. Pleasant, MI, USA
[*]kayas1a@cmich.edu

ABSTRACT

With the rise of streaming services like Amazon Prime and Netflix in recent years, movie recommender systems have not only become useful but paramount in helping us sort through an infinite number of movie choices. However, they have been focused mostly on individuals rather than groups. Moreover, existing techniques for movie recommender systems face the problem of having a static model that does not take into account the temporal effects of user-item interactions that can capture the evolving taste of a user. Recently, a more dynamic approach to collaborative filtering with recurrent neural networks (RNNs) has tackled this problem for individual recommendations. This paper builds upon the collaborative filtering with RNNs to solve the problem of recommending movies to groups. Furthermore, this solution also considers a novel group ranking algorithm that takes into account a rich set of factors available in the popular MovieLens dataset for the recommendations. This paper's RNN based approach is shown to slightly underperform another similar work[1] with top-10 categorical accuracy of 26% but achieve a better result with item coverage of 30%.

Introduction

Recommender systems have been massively utilized since the early days of the web to personalize recommendations for individuals. They can be a tool to help users find what they might like from an endless selection of items. The success of companies such as Amazon and Netflix can probably be attributed to their clever use of these personalized recommendations. With this progress, the next frontier of research in this area should have been group recommender systems. However, they have taken a backseat for various reasons despite their obvious usefulness in an activity like movie viewing. Consequently, most group recommender systems have relied on combining old approaches with only slight improvements. Fortunately, deep learning based approaches applied to recommender systems are becoming more popular, which is established by the abundant literature that is available on this topic[2–4].

Most of the research in the area of recommender systems for individuals has focused on a technique called collaborative filtering that is used for recommending items to users based on their past preference for items[5]. There are two main approaches to collaborative filtering: Matrix Factorization[6] and Neighborhood-based[7]. Matrix Factorization attempts to decompose the sparse user-item interactions matrix to extract latent factors from it. Whereas, Neighborhood-based approach either attempts to find similar users or similar items by calculating a similarity matrix. Moreover, these two approaches to collaborative filtering have been integrated to gain further performance improvements[8]. Although useful, in the real world, the popularity and perception of products change constantly, and customers' taste evolves. Recently, a more dynamic approach of collaborative filtering with RNNs has captured the temporal aspects of recommendations like evolving taste or context-dependent interests, which is showing promising results[1]. Yet this approach has not been explored for group recommender systems.

RNNs are a class of neural networks that are used to process a sequence and, therefore a powerful model for sequential data. They are different from other neural networks in that the property of remembering the past is inherently built into them. This is accomplished at a basic level by allowing previous outputs of the network to be used as inputs while having hidden states. However, there are more advanced types of RNN such as LSTM[9] (Long Short Term Memory) and GRU[10] (Gated Recurrent Unit) that have more sophisticated architectures. Recently, their success in speech recognition, translation, and other domains have made them popular for recommender systems[11,12].

The most prevalent methods used for solving the problem of group recommendations are user profile merging and recommendation aggregation[13]. User profile merging combines the ratings of individual group members to build a common user profile, which is then used to make recommendations for the group using collaborative filtering techniques[14]. On the other hand, recommendation aggregation generates recommendation lists for the individual group members and merges them to create the group's candidate list for ranking[15]. The ranking is usually done by calculating the group's relevance score for an item, which can be based on two strategies: average or least misery. Average strategy is selected for maximizing the average score of group members for an item. Whereas, least misery is selected to maximize the lowest score amongst all group members. A further improvement to the group's relevance score is a consensus function to balance an individual member's disagreements with the group's relevance[16]. While effective, these ranking functions suffer from relying too much on balancing the recommendations

for all the group members and can exclude items that a member would be interested in even though they may not rate them highly. Moreover, the ranking function is completely ignoring a rich set of factors such as popularity, interest, novelty, diversity, and freshness, which are quite important in producing a well-rounded group recommendation[17,18].

A different approach to movie recommender system for groups is required that capitalizes on the recent state-of-the-art deep learning techniques demonstrated by collaborative filtering with RNNs for generating the best group candidate recommendations. This approach has the benefit of taking the evolution of group members' tastes into account. Such a recommender system should also have a novel group ranking algorithm that takes advantage of the rich set of features available in the MovieLens dataset along with high group relevancy for the recommendations[17,18].

Methods

Data

One of the major constraints in finding datasets for sequence predictions is the availability of the order of events (in the form of timestamps). Fortunately, this information is available in the popular MovieLens dataset[19] and is the reason why this dataset was chosen. The MovieLens dataset[19] is a very popular and widely used dataset used for evaluating movie recommender systems from GroupLens, a research lab at the University of Minnesota. It was first released in 1998 and they have been curating people's preferences for movies ever since. The ratings were gathered from the MovieLens website[20], which provides personalized movie recommendations after entering your movie ratings. There are a variety of MovieLens datasets available ranging from 100K to 1B. For this particular work, the MovieLens 25M dataset[21] was chosen because of its recency and recommendation for new research.

The entire dataset consists of 25 million ratings applied to 59,000 movies by 162,000 users. The dataset was randomly split by users into training, validation, and test sets using a 60-20-20 split. To deal with the issue of fewer user ratings and less popular movies in the dataset, users and movies that had very few interactions were removed from the dataset. Since the problem is a sequence prediction problem for movies, users' sequence data had to be generated initially. This was accomplished by sorting the users' ratings data and generating the sequence of viewed movie IDs for each user. Furthermore, to ease the process of generating input sequence data, sequence files were generated for the training, validation, and test sets. These sequence files are then used in the mini-batch generator to generate the input sequence data of fixed batch size to train the model.

To encode the sequence data for training, an approach was adopted similar to the encoding performed when preparing data for learning the sequence of words[22]. This approach considers each movie rating as a word, the catalog of movies as the full vocabulary, and the history of each user as a sample sequence. At each time step, inputs are sequences of the one-hot encoded movies, and the target is a one-hot encoded subsequent movie. Although the MovieLens datasets have users' feedback in the form of ratings, they were not used for building the model and making predictions. Only the fact that a movie was rated or not by a user was used for this work.

Development Environment

The development environment used for this project was the Google Colab Pro Jupyter notebook environment supporting a TensorFlow and Keras stack. Most of the computations for larger networks were performed using NVIDIA Tesla v100-SXM2 graphical processing unit (GPU) hardware equipped with 16 GB of memory and 25.5 GB RAM.

Training and Evaluation Procedure

After preprocessing the MovieLens ratings data and the generation of sequence data, several RNN architectures were trained and evaluated. The initial architecture used a layer of vanilla GRU in all the experiments with 32 units and a masking layer by applying a mask value of 0 to skip timesteps when the sequence was less than the maximum length.

During training, the metric used to measure the performance was categorical cross-entropy for loss. The number of epochs was determined by monitoring the model performance on the validation data and stopping after the performance on the validation data started levelling off. Figure 1 illustrates the trajectory for training and validation top-10 accuracy for the best performing model. And, Figure 2 illustrates the trajectory for training and validation loss for the best performing model.

To optimize the performance of the model, several aspects of the RNN were adjusted. Additionally, optimizers such as Adam and RMSprop were primarily used because they worked particularly well for training this dataset. Table 1 summarizes the performance achieved by each architecture and it's configuration when training the model for 10 or 20 epochs to reach the optimal performance. It's important to note that due to a large number of hyperparameters, an exhaustive search of the hyperparameter space was not attempted.

The model built with the training set was evaluated on the first half of the sequence of movies watched by users in the test set. The evaluation metrics were then computed based upon the ability of the model to predict the next movie in the sequence and recommend several distinct movies. Several metrics were considered to gauge the performance of the model: top-k categorical

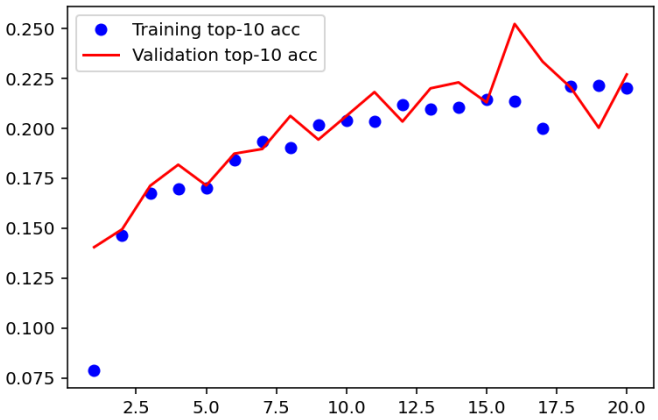

Figure 1. Training and validation top-10 accuracy

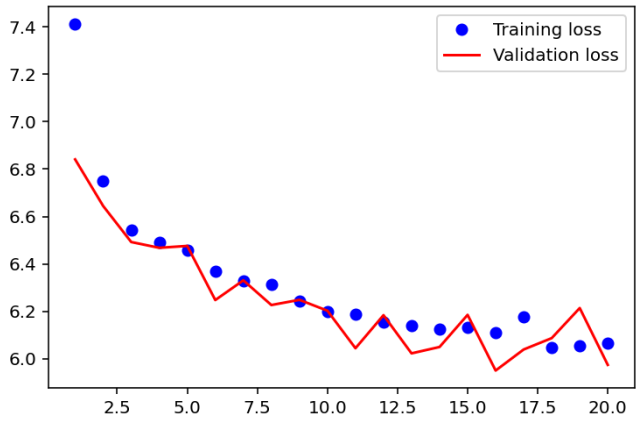

Figure 2. Training and validation loss

accuracy[2,3], item coverage[1], user coverage[1] and recommendations coverage. Top-k categorical accuracy gives a sense how often targets are in the top-k recommendations and calculates the percentage of correct movies in the top-k recommendations. Item coverage captures the ability of the method to recommend correct movies in the top-k recommendations and gives the total number of distinct movies that were correctly recommended. User coverage captures the ability of the method to make successful recommendations to users and computes the percentage of users who received at least one correct movie in the top-k recommendations. Recommendations coverage captures the ability of the method to recommend diverse movies in the top-k recommendations and counts the total number of distinct movies that were recommended. To have consistent metrics across the different architectures, they are computed "@10", i.e. the movie recommender system generates top-10 recommendations for each user.

With Adam as the optimizer, the training was very stable where training and validation top-k categorical accuracy increased after every epoch. Learning rate of 0.01 gave the best result for both Adam and RMSprop instead of the default learning rate. However, Adam was ultimately chosen for training all the other architectures because of its slightly better performance compared to RMSprop. Regarding the selection of data parameters while training the model, batch size of 128 was chosen to utilize the full capacity of the GPU and also because it slightly improved the performance of the model. In addition, maximum sequence length of 10 performed slightly better than 100 and 200. But the model's performance was on par irrespective of whether a shorter or longer sequence was chosen.

Although vanilla GRU was used in most of the architectures, other types of RNN were also evaluated to see if the additional complexity offered any further benefits. The RNNs that were explored were:

- **LSTM:** Long Short Term Memory RNN is a more complex version of GRU, with more gates and more parameters to tune. They were designed to avoid the long-term dependency problem, which allows them to remember information for long periods of time
- **2-Layered GRU:** This uses two GRU layers, one stacked (output of one layer feeding the next layer) over the another. The representational power of the network can sometimes increase with this approach.
- **Bidirectional GRU:** A Bidirectional GRU uses two GRU networks, where one reads the inputs in chronological order and other in anti-chronological order. The output of both is then merged and fed into the last output (softmax) layer. It exploits the order sensitivity of RNNs, increasing accuracy for some problems.

Results

In Table 1, it can be seen that the best performance across different vanilla GRU architectures on the test set was achieved with a configuration of 128 neurons, maximum sequence length of 10 and optimizer as Adam with learning rate of 0.01. When comparing LSTM and 2-Layered GRU with the same vanilla GRU configuration, they both slightly under-performed. However, Bidirectional GRU was able to achieve a boost in performance and was the best performing network architecture out of all. With this architecture, the model was able to achieve a top-10 categorical accuracy of 25.57% that is able to predict the next movie seen by approximately 26% of the users in test set across 10326 movies.

Given the sequential nature of the data, it was important to determine if using a complex approach such as RNNs for predicting top-k movie recommendations is valid. As a result, a naive top-k recommender was developed to produce recommendations based on a simple, common-sense approach for movies. If a user recently watched a movie, a common-sense approach is to always recommend the never seen top movies (highly rated on average) in the release year of the last watched movie. This naive top-k recommender achieved a top-10 categorical accuracy of 0.1%. These results are presented in Table 2.

Group Movie Recommendations

With the model tuned for individual recommendations, movie recommendations for groups are generated by first getting the list of individual top-k recommendations for each user in the group. The individual lists are combined to get all the unique recommendations for the group. Any movies in the group recommendations that have already been watched by any of the users are removed from the group list. The movies are then sorted and ranked according to the recency and popularity of the movies, which is the average ratings of the movies in the MovieLens ratings data. Finally, the top-k movies in the ranked list is recommended to the group.

Discussion

The performance of the approach presented here mirrors that of the similar work carried out by Devooght and Bersini[1]. They mostly used a vanilla LSTM network in their experiments. In their results, they were able to predict 33% of the next movie for top-10 recommendations on a holdout test set using only the sequence of movies[1]. Devooght and Bersini were able

Table 1. Comparison of different architectures' performance on MovieLens 25M test set across 10326 movies

Metrics (@10)				
Architecture	**Top-K Categorical Accuracy** (%)	**Item Coverage**	**User Coverage** (%)	**Recommendations Coverage**
GRU-Neurons(32)-Max Length(10)-[with Adam(lr=0.01)]	19.81	670	19.8	2530
GRU-Neurons(64)-Max Length(10)-[with Adam(lr=0.01)]	21.8	743	22.36	2837
GRU-Neurons(128)-Max Length(10)-[with Adam(lr=0.01)]	23.69	790	24.02	2956
GRU-Neurons(32)-Max Length(100)-[with Adam(lr=0.01)]	18.5	612	18.64	2179
GRU-Neurons(64)-Max Length(100)-[with Adam(lr=0.01)]	20.33	683	20.6	2414
GRU-Neurons(128)-Max Length(100)-[with Adam(lr=0.01)]	22.1	723	22.44	2622
GRU-Neurons(32)-Max Length(200)-[with Adam(lr=0.01)]	18.46	631	18.56	2145
GRU-Neurons(64)-Max Length(200)-[with Adam(lr=0.01)]	20.70	655	20.3	2492
GRU-Neurons(128)-Max Length(200)-[with Adam(lr=0.01)]	21.97	725	22.09	2626
GRU-Neurons(128)-Max Length(10)-[with RMSProp(lr=0.01)]	19.85	556	19.52	1311
GRU-Neurons(128)-Max Length(10)-[with Adam(lr=default)]	18.04	538	18.06	1679
LSTM-Neurons(128)-Max Length(10)-[with Adam(lr=0.01)]	22.84	741	22.84	2587
Bidirectional GRU-Neurons(128)-Max Length(10)-[with Adam(lr=0.01)]	25.57	812	25.26	3113
2 Layer GRU-Neurons(128)-Max Length(10)-[with Adam(lr=0.01)]	21.65	696	21.37	2627

Table 2. Performance of top-k recommendation methods on MovieLens 25M test set

Metrics (@10)		
Method	**Top-K Categorical Accuracy** (%)	**Item Coverage** (%)
Devooght and Bersini[1]	33*	18
Bidirectional Vanilla GRU RNN	26	30
Naive Top-K Recommender	0.1	-

* Note that values reported for Devooght and Bersini are from[1] and calculated from a different data set.

to achieve this higher accuracy because they introduced diversity bias in their model. Any model trained to optimize the success of predicting the correct recommendations will learn to recommend the most popular items, which are often useless recommendations. They achieved this by modifying the loss function (categorical cross-entropy) of their model by lowering the error associated with mispredicting the most popular items. This modification allows their model to recommend more rare items by counteracting the bias toward popular movies caused by the skewed distribution of popularity. On the other hand, the best RNN model presented in this paper was able to beat their item coverage of 18% with 30% suggesting that the introduction of diversity bias in their model might have sacrificed item coverage for top-10 categorical accuracy. However, it is important to note that their results were reported on MovieLens 1M dataset across 3706 movies.

Devooght and Bersini's[1] work also made use of the extra features in addition to the sequence of movies such as users' features, movies' features, interactions' features, and all features. Interestingly, the gain was significantly small even when all the extra features were combined, which suggests that the sequence of movies already contains most of the high-level information and making use of any additional features will only provide incremental improvements.

The value of recommending movies with sequence information utilizing RNNs are much better in terms of top-k categorical accuracy and item coverage because the best RNN model trained in this paper was able to predict the next movie with 26% accuracy for top-10 recommendations. Devooght and Bersini[1] have shown in their results that methods utilizing non-sequence information show relatively low top-k categorical accuracy and item coverage. Since, this paper's approach without the diversity bias has shown similar results with Devooght and Bersini's approach by including the diversity bias[1], it's quite likely that sequence information contains valuable temporal information that existing static approaches for recommendations such as collaborative filtering misses out on. An interesting finding in Table 1 was that using longer input sequences over shorter input sequences did not significantly degrade the performance of the model. However, shorter input sequences won in terms of the evaluated metrics, which suggests that the model is better at short-term predictions.

The input of the RNN model presented in this paper included only the sequence of movies (in the form of one-hot encoding of the last movie at each time step). However, MovieLens dataset possesses much richer information that might improve the model performance of the RNN model. Each additional feature can be one-hot encoded as before and concatenated to the normal input as an additional vector. With this concatenation, input of the RNN would be one-hot encoded sequence data plus any additional users' or movies' features. Another useful feature for training can be movies' tags and genre data that could be quite handy for predicting the user's favorite movie genres in a separate model output and using that as a filter to improve the recommendations.

The approach presented here was able to beat the baseline naive predictor by a factor of 260. The performance of the naive predictor shows that the preference for watching top movies in the release year of the recently viewed movie does not quite hold true statistically.

Conclusion

In this paper, a new approach for recommending movies using RNNs was explored, in particular, the GRU. Using RNNs is a more dynamic approach to collaborative filtering techniques that can take the evolving taste of a user into account. The experiments performed showed that GRU produces comparable results with a similar work[1] in terms of top-10 categorical accuracy and item coverage. This paper's approach slightly underperformed on top-10 categorical accuracy but was able to achieve a higher item coverage without the diversity bias. The performance of the approach for recommending movies on both short and long sequences was quite good, with short input sequences slightly beating long input sequences, thus showing the limitations of the approach for long-term predictions.

References

1. Devooght, R. & Bersini, H. Collaborative filtering with recurrent neural networks. *arXiv preprint arXiv:1608.07400* (2016).
2. Zhang, S., Yao, L., Sun, A. & Tay, Y. Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv. (CSUR)* **52**, 1–38 (2019).
3. Le, J. Recommendation system series part 1: An executive guide to building recommendation system. https://towardsdatascience.com/recommendation-system-series-part-1-an-executive-guide-to-building-recommendation-system-608f83e2630a (2019).
4. Le, J. Recommendation system series part 2: The 10 categories of deep recommendation systems that academic researchers should pay attention to. https://towardsdatascience.com/recommendation-system-series-part-2-the-10-categories-of-deep-recommendation-systems-that-189d60287b58 (2019).
5. Linden, G., Smith, B. & York, J. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* **7**, 76–80 (2003).
6. Koren, Y., Bell, R. & Volinsky, C. Matrix factorization techniques for recommender systems. *Computer* **42**, 30–37 (2009).
7. Aggarwal, C. C. Neighborhood-based collaborative filtering. In *Recommender systems*, 29–70 (Springer, 2016).
8. Koren, Y. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 426–434 (2008).
9. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural computation* **9**, 1735–1780 (1997).
10. Cho, K., Van Merriënboer, B., Bahdanau, D. & Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).
11. Graves, A., Mohamed, A.-r. & Hinton, G. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, 6645–6649 (IEEE, 2013).
12. Sutskever, I., Vinyals, O. & Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112 (2014).
13. Jameson, A. & Smyth, B. Recommendation to groups. In *The adaptive web*, 596–627 (Springer, 2007).
14. Yu, Z., Zhou, X., Hao, Y. & Gu, J. Tv program recommendation for multiple viewers based on user profile merging. *User modeling user-adapted interaction* **16**, 63–82 (2006).
15. O'connor, M., Cosley, D., Konstan, J. A. & Riedl, J. Polylens: a recommender system for groups of users. In *ECSCW 2001*, 199–218 (Springer, 2001).
16. Amer-Yahia, S., Roy, S. B., Chawlat, A., Das, G. & Yu, C. Group recommendation: Semantics and efficiency. *Proc. VLDB Endow.* **2**, 754–765 (2009).
17. Amatriain, X. & Basilico, J. Netflix recommendations: Beyond the 5 stars (part 1). https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429 (2012).
18. Amatriain, X. & Basilico, J. Netflix recommendations: Beyond the 5 stars (part 2). https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-2-d9b96aa399f5 (2012).
19. Harper, F. M. & Konstan, J. A. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* **5**, 1–19 (2015).
20. GroupLens. Movielens 25m dataset. https://movielens.org (2020).
21. GroupLens. Movielens 25m dataset. https://grouplens.org/datasets/movielens/25m/ (2020).
22. Kombrink, S., Mikolov, T., Karafiát, M. & Burget, L. Recurrent neural network based language modeling in meeting recognition. In *Twelfth annual conference of the international speech communication association* (2011).
23. Amatriain, X. & Basilico, J. Topkcategoricalaccuracy class. https://keras.io/api/metrics/accuracy_metrics/#topkcategoricalaccuracy-class (2020).