IMPROVED SESSION-BASED RECOMMENDATIONS USING RECURRENT NEURAL NETWORKS FOR MUSIC DISCOVERY

Yi Ye*, Yi Xie and Cheng Chen

College of Information Science and Technology, Jinan University, Guangzhou, Guangdong, 510000, China
* yeyi0@stu2016.jnu.edu.cn

Abstract. The number of songs on the music platform is large while users just have access to a fraction of songs. Recommendation systems are created to solve this problem and collaborative filtering is the most widely used method. But collaborative filtering methods are overly dependent on user ratings of projects. New recommendation methods, Deep Learning techniques, are proposed especially for items’ feature extraction and for session-based recommendation with RNN. In this paper, we considered a LSTM-based approach for session-based recommendation and built architecture for music discovery. This architecture is composed of four modules: MDM (Music Data Modeling), NSP (Next Song Prediction Using LSTM), MLB (Music Library Building) and R4U (Recommendation for Users). Improved session-parallel mini-batches and ranking loss function are leveraged by the architecture to modify the basic LSTM. Through the experiment, we obtained a relative improvement in ranking metrics (MRR@N and Recall@N) over the session-based algorithms baselines.

1. INTRODUCTION
Recommender Systems (RS) have been successfully applied to many fields, such as Netflix’s film [1], Facebook’s social network [19], enabling users to filter through a huge amount of information. Deep Learning, a hot topic in machine learning, has been increasingly popular in Recommender Systems (RS). Session-based recommendation using Deep Neural Networks, especially Recurrent Neural Networks (RNN), possess several properties that make the recommendation algorithm more efficient. The efficiency of RNN is reflected in its ability of modeling the sequence of user behavior and reflecting the interdependence between user behaviors.

In this paper, we try to build a recommendation architecture using LSTM, a type of RNN, based on the online music platform. There are three challenges for RS in music domain:

- Interpretability: It is difficult to judge the reason for recommending songs to users according to various factors and explain the accuracy of the recommended results through argumentation.
- Uniqueness: Different users have different preferences for music.
- Sequential forecasting: The user's demand for music is changeable throughout the time. Considering the change of aesthetic cognition, the current popularity of the era is also an important reason for users.

2. A DEEP LEARNING ARCHITECTURE FOR MUSIC RECOMMENDATIONS
As shown in figure 1, we built a large-scale deep learning-based music recommendation architecture
that consists of four modules: **MDM** (Music Data Modeling), **NSP** (Next Song Prediction), **MLB** (Music Library Building) and **R4U** (Recommendation for Users).

![Diagram](image.png)

**Figure 1**: A large-scale deep learning-based music recommendation architecture

### 2.1 MDM (MUSIC DATA MODELING)

This MDM model is responsible to extract features from lyrics of songs and metadata and to learn a distributed representation for each song. The inputs of this module are lyrics and metadata, and the outputs are sequences of song content embeddings. We select singers, albums and track’s duration as metadata and integrate them into metadata attributes. It’s empirical to use Continuous Bag-of-Words (CBOW), one of the word2vec algorithms, to pre-trained lyrics embedding. To create a word2vec model, we treat lyrics of the song as documentary. Next, the lyrics encoded by one-hot are fed to the word2vec model, where we use the CBOW algorithm. After the execution of CBOW, lyrics are transformed into a vector as a word embedding. Finally, these two pre-trained word embeddings, lyrics embeddings and metadata attributes, are concentrated by layer normalization technique and turn out to be a vector representing properties of the song.

### 2.2 NSP (NEXT SONG PREDICTION)

The task of the second module is to predict predicted next-song embedding. In this work, we only consider session-based information and ignore the user's past sessions due to the dynamic changes in user preferences and the high degree of sparsity of users. The inputs for second module are the song's popularity, newness and the Pre-Trained Song Content Embedding. The amount of listening information of all the songs in a day is retained, and the popularity is calculated by calculating the listening volume of a certain song. The newness is calculated in units of days by calculating the time difference between the song issuance date and the session. The above popularity and newness are caused by the log function smooth.

The Pre-Trained Song Content Embedding is in this case concatenated with popularity and newness to produce the user taste embedding. For a certain user $u$, we take User Taste Embedding of the last $n$ tracks that were played as input for purpose of getting a more accurate song recommendation and reflecting how the user's listening behavior changes over time.

This module uses a type of RNN - LSTM [10]. In this paper, we regard this task as a regression issue since our input is a real-value vector [3]. We hope that the training goal of LSTM is minimizing the distance between Predicted Next-Song Embedding and User-Taste Embedding (i.e., the Next Song Embedding, that is, the positive samples that are actually listened by the user in the session), that is, maximizing their similarity. Whilst maximize the Predicted Next-Song Embedding and the songs that are actually no listened by the user in the session, i.e. the distance between the negative samples, i.e. minimize their similarity.

We define the similarity between two embeddings as cosine similarity, as shown in the following formula:
\[ L(e_1, e_2) = \cos(e_1, e_2) = \frac{e_1 \cdot e_2}{\|e_1\|\|e_2\|} \]

For the efficiency of the calculation, we randomly select a certain number of negative samples in the set of all songs not listened by the users (negative samples), and the set of these negative samples is \( M' \). The set \( M' \) consists of \( M' \) and the set of User-Taste embedding of the next song listened by the user (positive sample). In this paper we use BPR (bayesian personalized ranking) \cite{18} as our ranking loss function. Here we define the score of for a given session as \( \hat{r}_{s,k} \) is the similarity between the Predicted Next-Song Embedding and sample \( k \). By comparing the positive sample, that is, the score of the User-Taste embedding \( \hat{r}_{s,i} \) and the average of the score \( \hat{r}_{s,j} \) of the negative sample, we can get a loss of the specified session function:

\[
l(\theta) = -\frac{1}{N_S \sum_{j \in M'} \log(\sigma(\hat{r}_{s,i} - \hat{r}_{s,j}))}
\]

Where, \( \theta \) represents the model parameters that need to be learned.

2.3 MLB (MUSIC LIBRARY BUILDING) & R4U (RECOMMENDATION FOR USERS)

The input to the third module is the song content embedding from the first module, and then we use spectral clustering \cite{23} to obtain the music categories of the songs and stores the results in the Music Library.

The input to the fourth module is the music categories obtained by the third module, the number of listeners for all songs in a day and the Predicted Next-Song Embedding for the second module. The pop and new of the songs in the database obtained by the third module are calculated based on the real-time data and produce a Live Song Content Embedding for each song in each class.

The Predicted Next-Song Embedding will be compared with the updated database classification results. First, search the center of the class closest to predicted Next-Song Embedding. Second, look for the top-N songs in this class that is most similar to Predicted Next-Song Embedding and recommend them to user. At the same time, the reason for the recommendation is presented to the user, that is, according to the user's historical data, we get some information - the style of the user's preference, the singer, the lyrics...

3. EXPERIMENTS

We use Tensorflow to implement our entire recommended process. The source of the neural network and baseline code is referenced to some open source code\(^1\). We crawled the data through the Netease Cloud Music API\(^2\).

3.1 DATA ANALYSIS AND FILTERING

Datasets need to be filtered and processed before they can be used as input for word2vec and RNN. Observing our data, we take it for granted as an NLP problem, that is, thinking of the user's playlist as documents, and the songs in the playlist as words. Referring to NLP's requirements for incoming data, we require data to meet the following criteria:

- Lyrics, artist, album and the track's duration information are complete.
- The length of the created playlist is \([1, 50]\), and the number of artists or albums corresponding to the songs in the playlist exceeds 1, which is for the diversity of prediction.

We do not need to limit the number of occurrences of the same song because a song can only appear once in a playlist. We screen 378, 2273 playlists, including 4, 237, 095 tracks, created by 33, 2427 users. We finally gather 200, 000 train sequences, 4, 000 validation sequences and 4, 000 test sequences due to the requirements of the model test.

\(^1\) one of the baselines: https://github.com/hidasib/GRU4Rec
\(^2\) https://github.com/Binaryify/NeteaseCloudMusicApi
3.2 TRAINING AND EVALUATION OF THE NSP

3.2.1 LSTM architecture
The neural network structure involved includes two recurrent layers and one dense layer. The number of input dimensions and output dimensions are $D_{in}, D_{out}$, respectively, and $D_{in} = D_{out}$, which is due to our forecast predicted next-song embedding in the same song space. In the recurrent layer, the hidden state dimension $D_{hid}$ is equal to 100. We use with a learning rate of $\alpha$, tanh activation function, and Adagrad [5] as a gradient-based optimizer in this work because it required to tune only one parameter. In the last layer, we use the linear activation function because we are predicting the original vector value.

3.2.2 Mini-batches
We naturally take User taste embedding of the last n tracks that were played as input. LSTM uses in-sequence mini-batches which is similar to natural language processing. In our architecture, sessions are represented by playlists and we take the number of times users listen to songs into account in the construction of the session [2]. We sorted the first n consecutive playlists created by the user in order of creation time and calculated their songs in the list. Building a play sequence can be expressed as follows: first, we arrange the order for all sessions; second, we use the first song of the first session as the input for the first mini-batch, and the second mini-batch consists of the second song, and so on. If the session ends, the location of the session is replaced with the next session. The session is independent, when this switch occurs, we reset the appropriate hidden state for the reason that the sessions are assumed to be independent. The specific input process is shown in the following figure:

![Figure 2: The specific input process](image)

3.2.3 Metric
We first choose Recall@N [14, 9], that is the proportion of cases having the desired song amongst the top-N songs in all test cases, to be our primary evaluation metric. The second metric used in the experiments is MRR@N (Mean Reciprocal Rank) [7]. The reciprocal of the standard answer in the results given by the evaluation system is taken as its accuracy. Therefore, MRR is sensitive to the position of clicked songs.

3.3 BASELINES
We used the following session-based algorithms as our baseline.

**POP**-Returns the most popular songs of all projects, the ones that have been listened to the most in the past day. This is always a strong baseline in a specific application.

**Item-KNN**- Always recommend the most similar songs to the last listened songs by comparing their cosine similarities.

**BPR-MF**- One of the commonly used matrix factorization methods. We use the average of item feature vectors of the items that had occurred in the session so far as the user feature vector. In other words, we average the similarities of the feature vectors between a recommendable item and the items of the session so far [7].
GRU4Rec - Seminal neural architecture using RNNs for session-based recommendations [4]. For this experiment, it was used GRU4Rec with BPR-max loss function, Adam optimizer (momentum = 0), learning rate of 1e-4 which is considered as the best architecture.

3.4 RESULTS
The parameters such as \{learning rate, size of mini-batch, \(D_{\text{in}}(D_{\text{out}})\}\} were optimized on a hold out validation set. Then the LSTM was retrained on the training set and the result was measured on the test set. Since the loss function \(l(\theta)\) is differentiable w.r.t to \(\theta\), the NSP module is trained using back-propagation on gradient-based numerical optimization algorithms.

In order to compare different learning rate and their results we use different parametrizations for NSP module. Figure 7 shows the influence of the Adagrad method when the size of mini-batch is 200 and \(D_{\text{in}}(D_{\text{out}}) = 500\).

![Figure 3: The influence of the Adagrad method when the size of mini-batch is 200 and \(D_{\text{in}}(D_{\text{out}}) = 500\).](image)

According to figure 3, learning rate \(\alpha\) around 0.05 gives the best result, i.e. recall@20 reach 0.670. And similar observations when do the same experiment for MRR@20.

Fix \(\alpha = 0.05\) and \(D_{\text{in}}(D_{\text{out}}) = 500\), and make a same experiment for the size of mini-batch; Then, fix \(\alpha = 0.05\) and the size of mini-batch=200 and make a same experiment for \(D_{\text{in}}(D_{\text{out}})\). Finally, we get the optimal combination of parameters: \(\alpha = 0.05\), \(D_{\text{in}}(D_{\text{out}}) = 500\), size of mini-batch=200.

Through experiments, we compare our work with the baseline and get their performance on two indicators: Recall@N and MRR@N. The results of the experiments are shown in Table 2. In bold we mark the best performing model for each task. Obviously, GRU4REC and Item-KNN are the two best performers in the baseline. The performance of our work is similar to that of GRU4REC but slightly better. On the Recall@N indicator, our work performed well, which is more profitable than Item-KNN and slightly better than GRU4REC (except Recall@ 50). On the MRR@N indicator, our work also performed well, outperforming all baselines on MRR@10, but the results on MRR@20 and MRR@50 were lost to item-KNN.

It is worth mentioning that we have introduced the music library database, which has greatly shortened our training and testing time and does a great help to practical applications.
Table 1: The results of the experiments on two indicators: Recall@N and MRR@N.

| Training method | Recall@N   | MRR@N    |
|-----------------|-----------|----------|
|                 | N = 10    | N = 20   | N = 50   | N = 10    | N = 20   | N = 50   |
| POP             | 0.0071    | 0.0192   | 0.0523   | 0.0013    | 0.0032   | 0.0135   |
| Item-KNN        | 0.4958    | 0.5176   | 0.5339   | 0.2004    | 0.3421   | 0.3638   |
| BPR-MF          | 0.0832    | 0.1673   | 0.2665   | 0.018     | 0.0224   | 0.0481   |
| GRU4Rec         | 0.6374    | 0.6547   | **0.6422** | 0.2367   | 0.2539   | 0.2414   |
| **Our work**    | **0.6533** (+27.08%) | **0.6701** (+25.22%) | **0.6392** (+24.65%) | **0.2512** (-8.94%) | **0.3029** (+23.74%) | **0.2405** (+20.33%) |

4. RELATED WORK

The collaborative filtering method is the most widely used recommendation and it was first proposed by Goldberg, which greatly promoted the development of recommendation system in the 1990s [6]. The basis of modern collaborative filtering lies in latent-factor models, such as matrix factorization [12]. Some other matrix factorization techniques have been proposed to fit for explicit feedback settings [11,13,16].

Deep learning has achieved great success in learning item features from unstructured data, such as text, music and images. At early stages, neural networks were applied to RS through using a two-layer RBM in matrix factorization [20]. The first Deep Learning for Recommender Systems workshop at the ACM RecSys 2016 applied RNN to the Session-based Recommendation, and designed the RNN training, evaluation method and ranking loss for this task for the first time [7]. Session-based recommendation began to draw people’s attention. Subsequent research is based on research by Balázs Hidasi et al [8, 17, 22]. For example, explore how to add item attribute information (such as text and images) to the RNN framework to explore several model frameworks that fuse item attributes [8].

5. CONCLUSION

In this study, we proposed a recommender architecture for music discovery, using Word2Vec to extract textual features from songs’ lyric and a LSTM layer to model the playlist built by users in user sessions. We modified the basic LSTM in order to fit the task better by introducing improved session-parallel mini-batches and ranking loss function. We showed that our method can mostly outperform popular baselines, which can improve that our work can used in both deep learning applications in recommender systems and session-based recommendations in general.

ACKNOWLEDGE

The authors would like to thank 2018 China’s National College Students' innovation and entrepreneurship training program for providing financial support. At the same time, the authors are very grateful to the instructor Suohai Fan for his careful guidance.

REFERENCES

[1] Bennett, J., & Lanning, S. (2007). The netflix prize. In: Proceedings of KDD cup and workshop. San Jose. pp. 35.
[2] Bogina, V., & Kuflik, T. (2017). Incorporating Dwell Time in Session-Based Recommendations with Recurrent Neural Networks. In: RecTemp@ RecSys. Como. pp. 57-59.
[3] De Boom, C., Agrawal, R., Hansen, S., Kumar, E., Yon, R., Chen, C. W., ... & Dhoedt, B. (2018). Large-scale user modeling with recurrent neural networks for music discovery on multiple time scales. Multimedia Tools and Applications, 77(12): 15385-15407.
[4] de Souza Pereira Moreira, G., Ferreira, F., & da Cunha, A. M. (2018). News Session-Based Recommendations using Deep Neural Networks. In: Proceedings of the 3rd Workshop on...
Deep Learning for Recommender Systems. Vancouver. pp. 15-23.

[5] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(7): 257-269.

[6] Goldberg, D., Nichols, D., Oki, B. M., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. Communications of the ACM, 35(12): 61-71.

[7] Hidasi, B., Karatzoglou, A., Baltrunas, L., & Tikk, D. (2015). Session-based recommendations with recurrent neural networks. arXiv preprint arXiv:1511.06939.

[8] Hidasi, B., Quadrana, M., Karatzoglou, A., & Tikk, D. (2016). Parallel recurrent neural network architectures for feature-rich session-based recommendations. In: Proceedings of the 10th ACM Conference on Recommender Systems. Boston. pp. 241-248.

[9] Hidasi, B., & Tikk, D. (2012). Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Bristol. pp. 67-82.

[10] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8): 1735-1780.

[11] Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative Filtering for Implicit Feedback Datasets. In: Proceedings of the 8th IEEE International Conference on Data Mining (ICDM). Pisa. pp. 263-272.

[12] Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, 8: 30-37.

[13] Lee, D. D., & Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In: Advances in neural information processing systems. Vancouver. pp. 556-562.

[14] Liu, Q., Chen, T., Cai, J., & Yu, D. (2012). Enlister: baidu's recommender system for the biggest chinese Q&A website. In: Proceedings of the 6th ACM conference on Recommender systems. Dublin. pp. 285-288.

[15] Pan, R., Zhou, Y., Cao, B., Liu, N. N., Lukose, R., Scholz, M., & Yang, Q. (2008). One-class collaborative filtering. In: 2008 Eighth IEEE International Conference on Data Mining. Pisa. pp. 502-511.

[16] Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. In Proceedings of KDD cup and workshop. San Jose. pp. 5-8.

[17] Quadrana, M., Karatzoglou, A., Hidasi, B., & Cremonesi, P. (2017). Personalizing session-based recommendations with hierarchical recurrent neural networks. In: Proceedings of the 11th ACM Conference on Recommender Systems. Como. pp. 130-137.

[18] Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2009). BPR: Bayesian personalized ranking from implicit feedback. In: Proceedings of the 25th conference on uncertainty in artificial intelligence. Montreal. pp. 452-461.

[19] Resnick, P., & Varian, H. R. (1997). Recommender systems. Communications of the ACM, 40(3): 56-59.

[20] Salakhutdinov, R., Mnih, A., & Hinton, G. (2007). Restricted Boltzmann machines for collaborative filtering. In: Proceedings of the 24th international conference on Machine learning. Corvalis. pp. 791-798.

[21] Steck, H. (2015). Gaussian ranking by matrix factorization. In: Proceedings of the 9th ACM Conference on Recommender Systems. Vienna. pp. 115-122.

[22] Tan, Y. K., Xu, X., & Liu, Y. (2016). Improved recurrent neural networks for session-based recommendations. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems. Boston. pp. 17-22.

[23] Von Luxburg, U. (2007) A tutorial on spectral clustering. Statistics and computing, 17(4): 395-416.