Could Transformer Outperform Long Short-Term Memory Model in Predicting User Ratings?

Course project for CSE 6240: Web Search and Text Mining, Spring 2022

Lixing Liu
lliu374@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Jiarui Xu
jxu605@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Xingyu Li
xingyu@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Han Bao
hbao34@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

ABSTRACT

The project aims to find out whether the recently popular attention-based model of Transformer [8] could perform better than previous baseline proposed in the work by He et al. [4]. We will compare the performance of transformer model using similar input setup to the work of Behavior Sequence Transformer [1] to a baseline Long Short-Term Memory model using the same input setup [5]. We measure the performance of the two models on the dataset MovieLens [3] that contains 3,900 movies, 6,040 users, and 1,000,209 ratings, and each user has rated at least 20 movies, with integer scores ranging from 1 to 5. MovieLens is widely used to test and develop recommendation algorithms, especially collaborative filtering algorithms. The two models are trained and validated on the task of predicting user ratings based on sequences of feature inputs. The Long Short-Term memory model achieves an average root mean square error of approximately 1.3547 on validation data, and the transformer model achieves an average root mean square error of approximately 1.2122 on validation data.

1 INTRODUCTION

1.1 Aim

We're trying to figure out whether the Transformer model [8] could do better than plain Long Short-Term Memory model [5] by measuring how close the models could predict user ratings based on sequences of feature inputs including user attributes such as age, sex, occupation, and movie attributes, such as genre extracted from the MoiveLens [3] dataset.

1.2 Challenges

Most current recommendation models have significantly low accuracy. For Item-Item Collaborative Filtering model, to the lack of data, it is difficult to propose users or items with similar features [7]. For User-based Collaborative Filtering Recommendation model [6], it is hard to evaluate its trust level. Neural Collaborative Filtering [4] model has a Normalized Discounted Cumulative Gain of only 0.447, which indicates that the actually interacted item is not very likely to be ranked highly among predicted items, suggesting comparatively low accuracy of the model [4].

1.3 Impact

This project explores the use of the Transformer model [8] in a recommendation system. The attention layers in the Transformer model is proven to improve the success of many models in recent years, and it has become omnipresent in many state-of-the-art machine learning models. Therefore, it is important for us to examine if the transformer model [8] with multi-head attention can have higher performance in rating prediction tasks than existing recurrent models, such as the Long Short-Term Memory model [5]. The result demonstrates that the transformer model could generate closer rating prediction to the ground truth rating than the Long Short-Term Memory model. In recommendation systems, even a slightly increase in prediction accuracy can lead to an increase in profit margin, and therefore, we conclude that this project can lead to an increase in profit margin of current recommendation systems. Proven that the transformer model can have a higher recommendation accuracy, it is also worthy for future researches to further explore the potential of the transformer model, and to build a better recommendation system for sequential prediction based on our results.

2 LITERATURE SURVEY AND BASELINES

2.1 Literature Survey

MovieLens [3] is one of the most popular datasets in the world. Many research use these database to test and develop their core algorithmic advances in recommender systems, including:

Item–Item Collaborative Filtering
In 2001, Badrul Sarwar et al. [7] came up with a new algorithm-Item-Based Collaborative Filtering Recommendation Algorithms, which used MovieLens as their training and test dataset. However, in many systems, the number of users and items is very large, and the interaction information between users and items is often very small, which leads to the sparse user-item matrix. Due to the lack of data, it is difficult to propose users or items with similar features [7].

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
2018. ACM ISBN 978-1-4503-XXXX-X/18/06...$15.00
https://doi.org/XXXXXXX.XXXXXXX

User-based Collaborative Filtering Recommendation Massa and Avesani [6] proposed the User-based collaborative filtering recommendation algorithm. It uses statistical techniques to find neighbors with the same preferences as the target and then generates recommendations to the users according to the preferences of the neighbors. The main challenge of this algorithm is low scalability. The algorithms require computation that grows with both the number of users and the number of items, which means the it gets bigger with time.

2.2 Baseline: Long Short-Term Memory Model

Long Short-Term Memory model [5] can learn to minimize time lags over 1000 discrete time steps by forcing a constant error flow through a constant error carousel within a particular unit. The multiplication gate unit learns to open and close into a constant stream of errors. Long Short-Term Memory model is local in time and space. The computational complexity and weight of each time step are O(1). According to the original authors, Long Short-Term Memory is more successful and learns faster than previous methods such as real-time repetitive learning, time back propagation, repetitive cascade correlation, Elman networks, and neural sequence chunking. Long Short-Term Memory also solves complex, artificial, long-lag tasks that previous recursive network algorithms have never solved. Since Long Short-Term Memory model has good ability in processing long sequences of data representations, it has been widely used in recommendation systems which involves sequential prediction of user interaction based on input features. Besides, Devooght et al. has previously applied the model to the MovieLens dataset [2].

2.3 Transformer

Encoder and Decoder Stacks The encoder is composed of a stack of $N = 6$ identical layers. The decoder is also composed of a stack of $N = 6$ identical layers.

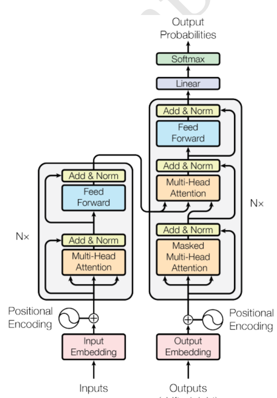

Figure 1: Transformer framework [8]

Attention function

(1) Scaled Dot-Product Attention

The input consists of queries and keys of dimension d_k, and values of dimension d_v. Scientists compute the dot products of the query with all keys, divide each by $\sqrt{dk}$, and apply a softmax function to obtain the weights on the values.

(2) Multi-Head Attention

Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, the transformer model is beneficial to linearly project the queries, keys, and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys, and values we will then perform the attention function in parallel, yielding d_v-dimensional output values.

Position-wise Feed-Forward Networks Each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically.

Embeddings and Softmax Similar to other sequence transduction models, scientists use learned embeddings to convert the input tokens and output tokens to vectors of dimension.

Positional Encoding To make use of the order of the sequence due to lack of convolution layers or sequential layers, scientists inject some information about the relative or absolute position of the tokens in the sequence.

Shortcoming The model is originally developed for sequence to sequence translation, so it requires constructing users' interactions as sequences. But this shortcoming would be out-weighted by its ability to extract good features from data.

3 DATASET DESCRIPTION

3.1 Dataset summary

The dataset contains a rating table which contains about 1 million ratings data and each of the rating is given by a user for a movie at a timestamp. The dataset was created by about 6400 users. It contains 1,000,209 ratings across about 3900 movies.

Table 1: Dataset Summary

Dataset	Movies	Users	Interactions
MovieLens	3900	6040	1,000,209

3.2 Data preparation

3.2.1 Source.
Our project will use the Movielens dataset from https://grouplens.org/datasets/movielens.

3.2.2 Data preprocessing steps and explanations.
For features in Movielens:

(1) The movies that appear first in search results for a user are those movies that the algorithm predicts that the user will

rate the highest. Movielens version4 blends a popularity factor in with predicted rating to order the recommendation lists.

(2) Tags appear in the later MovieLens datasets (10M and 20M). This feature allows users to apply tags — words or short phrases — to movies. MovieLens displays tags next to movies. Tags are clickable to show a list of movies on which that tag has been applied.

The visibility and ordering of tags differed by the user in early versions to provide A/B testing data, but by 2006 the interface was consolidated, sorting tags by the likelihood that the tag is "high value" according to a published metric. In January 2007, MovieLens launched a tag rating feature that put clickable thumbs-up-and-down icons next to tags. In Spring 2009, the tagging interface was given a new feature called "tag expressions" that dramatically influenced tagging behavior. This interface allowed users to retag movies much more easily and caused an increase in the rate of tagging activity, as well as an increase in tag diversity.

To preprocess:

(1) **Feature selection** The ratings data are only pairs of user_id and movie_id and the corresponding ratings. To make our prediction more robust, we select additional user features from the user table and additional movie features from the movie table. The additional user features are: sex, age_group, and occupation. There are two possible values for sex: "M" for male and "F" for female. There are seven age groups: under 18, 18-24, 25-34, 35-44, 45-49, 50-55, 56+. Each age group is represented using a single number: 1 for under 18, 18 for 18-24, 25 for 25-34, 35 for 35-44, 45 for 45-49, 50 for 50-55, 56 for 56+. There are 21 kinds of possible occupations, 20 of them have a specific name, such as "artist", and 1 is for "others". The additional feature for movies is genre. There are 18 kinds of genres, such as "Action" and "Adventure". A movie may have more than one genre.

Table 2: Selected Features

Feature Name	Example Value	Number of possible values
user_id	1024	3883
user: sex	"M"	2
user: age_group	18	7
user: occupation	"artist"	21
movie_id	1024	6040
movie: genre	"Action"	18

(2) **Feature pre-processing** We pre-processed the feature values, such that each unique value in each feature category other than user_id and movie_id would get a unique integer identifier, or "index". The "index" would be continuous and range from 0 to the number of possible unique values. For example, sex would turn into 0 and 1 after pre-processing. The first occupation, "artist", would get an "index" of 0, and the last occupation, "writer", would get an "index" of 20 after pre-processing. The second age group 18-25 (represented by 18) would get an "index" of 1 after pre-processing. Processing movie genres is more complex, since a movie may have more than one genre. We use a one hot encoding to represent whether a genre is an attribute of a movie. Each movie would have a corresponding feature vector of length 18 (the total number of genres), and the corresponding bit will be 1 if the movie is that certain genre, and vice versa. After the pre-processing, all feature representations would be in non-negative integer numbers.

(3) **Sequence generation** We join the pre-processed user and movie features table with the ratings table. We group the ratings by users and then sort each rating group by timestamp. Then we cut the sequence of ratings for a single user into smaller sequences of length 10. If there not enough data to make a sequence of 10, then the sequence is ignored. During training, if we set the batch size to 10, 10 such sequences would be loaded. We shuffles the order of sequences before training and evaluation. The contents in each sequence remains unchanged.

4 EXPERIMENTAL SETTINGS

Our task will be a regression task since we are predicting the rating of a user for a specific movie given sequences of length 10 of feature inputs (movie_id, user_id, age, sex, occupation, genre). Each sequence represents a sequence of movies rated by the same user sorted in by timestamp, and the input features user_id, age, sex and occupation are related the users and input features movie_id and genre are related to the movies. 85% of the sequences are used as the training data, and 15% of the sequences are used as validation and testing data. Each training and testing batch would have 10 of such sequences, so batch size is 10. We will use root mean square error to train the model and use average root means square error over all validation data for evaluation since the root mean square error measures the deviation of the predicted rating from the ground truth ratings, and is suitable for regression tasks. The random seed is set to 0 so the results could be reproduced. The system settings are as follows: CPU is Intel Xeon CPU with 2.2GHz, GPU is Nvidia Tesla T4 GPU of 16GB memory.

5 METHOD

(Option 1) Running existing methods

The first method we use is Long Short-Term Memory model [5]. We use the official implementation of the model by PyTorch, and the documentation of the programming interface could be accessed at the following link https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html. Long Short-Term Memory model has been widely used as the building block of many recurrent deep neural networks. The memory cell simulates the long-term memory of human beings, and the multiple gates connecting the input, previous hidden state and memory cell state to generate the next hidden state, memory cell state and output. In this project, we input sequences of features to the model to get a rating prediction for each time step in the sequence. Each time step consist of 6 features: user_id, movie_id, sex, age, occupation, genre. All time steps in the sequence share the same user and therefore the user-related features: sex, age, occupation. The sequence represents a continuous sequence of rating of movies generated by a single user sorted by the original timestamps. Each sequence is of length 10. For hyper-parameters, we use 3 layers and a hidden dimension of 16 for Long Short-Term Memory with drop out probability of 0.1. The learning rate is 0.01, and the batch size is 10. We use the Long Short-Term Memory model since the model is a widely used building block in many other models and is also suitable for a sequential prediction task.

The other method we use is based on the architecture of the Transformer model [8]. We use the official implementation of transformer by PyTorch, and the documentation of the programming interface could be accessed at the following link https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html. We uses pass data to the transformer in the way similar to that used in Behavior Sequence Transformer [1], where they input the same feature sequences to both the "Inputs" and "Outputs" component of the transformer networks. In this way, the multi-head attention layer which connects the "Inputs" and "Outputs" component would generate the self-attention result of the feature sequences. (Please refer to figure 1 for transformer model architecture) The feature sequences has the exact same format as the inputs to the Long Short-Term Memory model. The key advantage of the transformer model is the application of multi-head attention to generating attention output from embedding of the features. Each transformer block is built upon multiple linear layers and the attention layers, and the model consists of multiple transformer blocks. As transformer takes sequential data, we will use sequence of user interaction with movies to predict the sequence of rating outputs. For hyper-parameters, we use learning rate of 0.01, batch size of 10, hidden dimension of 16, 2 attention heads, 3 layers of encoders and 3 layers of decoders. We selected this method as the comparison to the baseline method since the transformer model has proven its ability to extract more context-aware representation [8] and could possibly perform better than the baseline method.

6 EXPERIMENTS AND RESULTS

We ran the training for 10 epochs for each model. The following table shows the result of the two model measured in Root Means Square Error (RMSE) running on validation dataset after 10 epochs of training with learning rate of 0.01 and Stochastic Gradient Descent optimization algorithm.

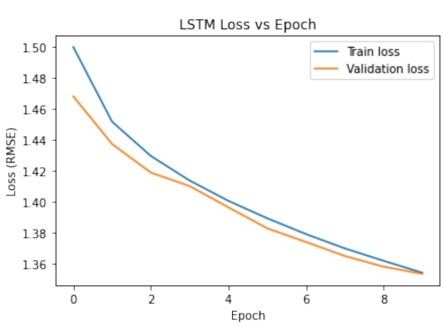

Figure 2: Long Short-Term Memory [5] Loss Graph

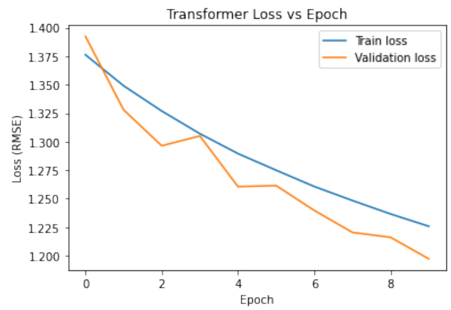

Figure 3: Transformer [8] Loss Graph

The above graphs display the training loss of the two models respectively during training. It shows that Transformer model [8] has lower loss than the Long Short-Term Memory model [5] in each epoch. The graph shows that for Long Short-Term Memory model, the loss is descending slower towards the end, meaning that it would be harder to continue optimizing the model. However, the loss of Transformer model is still descending comparatively quickly meaning that the model could be further optimized with more epochs. If we have trained more epochs, we would see transformer outperforming even more.

Table 3: Testing results

Metric	Transformer	Long Short-Term Memory
RMSE	**1.2122**	1.3547

As shown in the table shows that the transformer model outperforms the Long Short-Term Memory model in the task of predicting users' rating on movies given the same sequence of feature input for having a smaller average Root Mean Square Error. There are several potential explanations for this result:

(1) Transformer is better at capturing time sequence relationship without the problem of gradient explosion or vanishing in recurrent networks since transformer eliminates recurrent structure but rather uses positional encoder to represent the sequential information.
(2) The attention layer generates better weighting of the input features across timestamps than models using recurrent structure, and the weighting would help the model find out the importance of each feature.
(3) There are simply more parameters in the transformer model since there are more linear layers in the attention layers and also the feed-forward layers, and usually more layers and parameters suggests better performance since the model would be better capturing more subtle features.

As a result, it is not surprising that the transformer model outperforms the Long Short-Term Memory model as the result shows.

7 CONCLUSION

Our results demonstrate that Transformer model [8] generally performs better than the Long Short-Term Memory model [5] on MovieLens dataset [3]. However, the comparison is not comprehensive enough. There are other more advanced model than simply Long Short-Term Memory model, and we need to compare more of such models to the transformer model if possible. Also, the MovieLens dataset [3] is comparatively limited since the categories of features are small and interaction data is not that diverse.

To further understand the performance of the Transformer model [8], future works should evaluate the performance of the transformer model on more variety of data and compare against more kinds of model in sequential prediction tasks in the field of recommendation systems. Proven that the transformer model can have a higher recommendation accuracy, it is worthy to further explore the potential of the Transformer model [8], and to build a better recommendation system for sequential prediction.

8 CONTRIBUTION

All team members have contributed a similar amount of effort.

REFERENCES

[1] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou. 2019. Behavior Sequence Transformer for E-commerce Recommendation in Alibaba. *CoRR* abs/1905.06874 (2019). arXiv:1905.06874 http://arxiv.org/abs/1905.06874

[2] Robin Devooght and Hugues Bersini. 2016. Collaborative Filtering with Recurrent Neural Networks. *CoRR* abs/1608.07400 (2016). arXiv:1608.07400 http://arxiv.org/abs/1608.07400

[3] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article 19 (dec 2015), 19 pages. https://doi.org/10.1145/2827872

[4] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web* (Perth, Australia) *(WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 173–182. https://doi.org/10.1145/3038912.3052569

[5] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[6] Paolo Massa and Paolo Avesani. 2007. Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems.* 17–24.

[7] Badrul Sarwar, 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.* 285–295.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf