Sequential Sessions Based Music Recommendations Using Deep Learning

Ahmad Khan
akhan361@gatech.edu

Tianyuan Cui
tcui33@gatech.edu

Sagar Arora
sarora83@gatech.edu

Abstract

We use ideas from recent Deep Learning research in Natural Language Processing (NLP) and Deep Reinforcement Learning (DRL) to see if we can effectively model the user sequence prediction problem using user sessions data from music streaming platform Spotify. We focus on two tasks. First, in the "track sequence recommendation" problem we use the tracks played in the first half of the sessions to predict the sequence of tracks a user would play next in their remaining second half. Second, in the "skip (engagement) prediction" problem, we aim to predict if user will skip specific tracks in the second half of their session. We use LSTM and Transformer based encoder-decoder architectures and compare these approaches with a modified DDPG actor-critic architecture based on Reinforcement Learning. Our results based off 7 different sequential models show some promising performance on each of the two different tasks.

1. Introduction

Many deep learning recommendation systems currently utilize sequential user dynamics, using user session data, to capture some context of user activity in order to make recommendations to users. However the traditional sequential recommender system literature has focused mostly on modeling the next item "click" a user may have given their sequence of actions leading up to that click. For example Hidasi et al [7], use GRU recurrent units to model user sequential session data to predict their next session action while Kang et al [8] use Transformers and Sun et al [11] extend BERT for the same task of predicting the next session click. In Deep Reinforcement Learning, Zhao et al [14] made the first implementation of DDPG [9] within the context of a recommendation system. Much less literature (if any) however exists on utilizing sequential user session data to further predict an entire sequence of actions a user may take and not just the next click, given their sequential activity.

In this paper we look to address this more challenging problem to predict not just the next click but the entire next sequence of actions a user may take up to a predetermined t time steps in the future. We do this by solving two problems using the Spotify Skip Prediction Challenge Data [1]. Spotify is a large online music streaming platform on which users can listen to music. We are provided with sequential logs of more than 3.3M user listening sessions of length 20 each where each item in a session corresponds to a track (song) a user played along with a boolean indicator if they "skipped" the song or not. In addition we are also provided with 26 numerical features about the song like its "acousticness" etc but for legal reasons we are not provided any metadata about the song including song name, genre, artist etc. In our sampled dataset we have 103,000 unique track ids each corresponding to a unique song.

For the first problem, which we call the "track sequence recommendation" problem, we aim to predict the entire sequence of the next 10 tracks a user would play in their remaining session given the first 10 songs they listened to. We split the session in half with first 10 sequence of tracks serving as input while the latter sequence of 10 tracks is what we predict.

For the second sequence prediction problem, which we call the "track skip (engagement) prediction" probleem, we aim to predict which tracks a user skips in the second half of their session given session sequence information from their first half. A skip here is defined as a user clicking to skip a song from being played further within the first 1/3 duration.

For example, a user listens to track id sequence S = $\{1, 10...5, 43\}$ and is next going to listen to sequence T = $\{2, 9...34, 23\}$. For the first track sequence recommendation problem we try to correctly predict the next sequence of songs T. For the second skip prediction problem we now sequentially predict for each element of T, if a user will skip it or not given the input sequence S.

We believe solving these problems is important because first predicting a longer sequence a user interacts with beyond the next item a user may click gives businesses much greater value and ability to personalize how they recommend items to users. Second, by predicting "skips" we can think of it as predicting a stronger signal of engagement on the user's part than just viewing or listening. So by using both these problems in a hybrid fashion a business can first predict the sequence of next K items a user will be interested in and then also if they will "engage" or

not. Furthermore, by modeling two different sequence prediction problems our goal is also to highlight and contrast the difference in modeling performance we observe comparing one simpler sequence prediction problem that only involves predicting a sequence of boolean indicators (skip prediction) vs a highly complex problem that involves recommending a sequence of items from a much larger space of 103K possible options.

2. Approach

While there is currently limited research in recommendation systems to predict entire sequences of user activity, recent research in NLP involving sequential text has focused on natural language modeling tasks like Language Modeling and Neural Machine Translation which involve predicting entire text sequences. Therefore, due to the sequential nature of both our input as well as the output for both problems we utilize a traditional NLP Encoder-Decoder [12] architecture typically used for Machine Translation where we encode the first half of the session and thereafter decode the second half sequentially (figure1). For the track sequence recommendation problem, the input to the Encoder is the input sequence of 10 track ids played in the session so far which are mapped to a learned track embedding of 103K vocabulary size (each index corresponding to an individual track). This embedded input is encoded and passed to the decoder whose output is then passed to a Softmax output layer of 103K size to generate probabilities for each 103K next possible tracks. During training in the Decoding step, we right shift our true track sequence labels (to avoid peeking) and utilize teacher forcing by passing the actual song played during the last time step t as input for next time step t+1. However during evaluation we use the Greedy Decoding mechanism where we take the track id with the highest soft-max score as the input track id token into the next input of our predicted sequence, repeating until we reach the end.

For the "skip (engagement) prediction" task we follow a similar architecture but with a few modifications. First the output layer is a boolean 1/0 2D softmax indicating skip/no skip of the track instead of 103K possible track probability scores. Second, in the input sequence we also pass in the skip information of whether a track was skipped or not by masking the track-id of the skipped input tracks with a special $< SKIP >$ token. Our reasoning for doing so is that this gives us some information about a user's prior "skipping behavior" (e.g: they always skip songs) and also non skipped tracks could potentially reveal the user's intent better than skipped tracks. Finally, skipped tracks in general might have similar kind of characteristics (e.g: not good songs). For skip prediction decoding we assume we are given the true sequence of next track ids and so only predict the sequence of user skip actions for remaining songs.

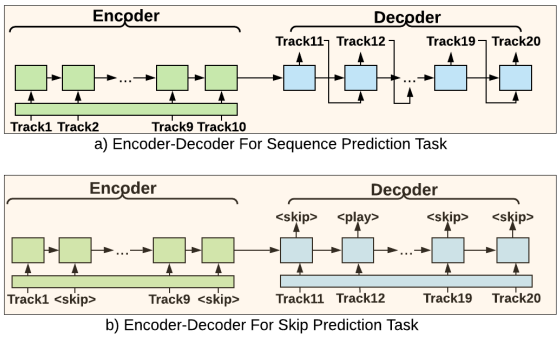


Figure 1: Encoder-Decoder Architecture Overview

2.1. RNN Based Model Architectures

Within our Encoder-Decoder framework we first try RNN methods using LSTMs or GRU units. We tried three different variants for the skip prediction problem.

2.1.1 Standard LSTM

We use LSTM in both the encoder and decoder with one 512 dimensional hidden layer for both. We utilize LSTMs as they have been quite effective in modeling sequential tasks since they utilize a hidden state to keep track of what has been observed previously.

2.1.2 LSTM-Word2Vec

Here we modify the above by pre-training word2vec based embeddings for tracks. Word2vec based embeddings can capture track similarity well and assist in downstream tasks. For pre-training we consider each session as a document and all played tracks as words in the document. We train the word2vec model (window size 10) on this corpus to learn a unique track embedding. When solving our sequence prediction problems we then initialize our weights for the track embedding with this word2vec learned embedding. Our total embedding size is 126 (26 dimensional track embedding concatenated with 100 dimensional word2vec embedding)

2.1.3 Bidirectional GRU

Same as 2.1.2, but with LSTM unit replaced by a GRU unit (and Bidirectional for Encoder). Our hypothesis for doing so was that bidirectionality could better capture relationships both ways in a sequence and GRU units may be less prone to forget long term dependencies.

2.2. Transformer Based Architectures

However, the above RNN methods can have some shortcomings. Even though LSTM/GRU help alleviate the vanishing gradient problem they can still struggle learning very

long-term dependencies. And even though using Bidirectionality during Encoding can help with generating contexts that look at both sequences before and after an item this may not be as effective as simply weighting entire sequences at once using attention mechanisms. Therefore we try the Transformer architecture next (Vaswani et al [13])

2.2.1 Standard Transformer Encoder-Decoder

We modify our Encoder-Decoder framework in Figure 1 to process inputs all at once and we utilize self attention and multi head attention on these to generate an encoded representation of the song sequence much like [13]. Unlike RNN methods, we also pass a learned positional embedding to capture some notion of ordering in sequence. For Decoding, we use the same Masked Transformer Decoder architecture mentioned in [13] where we mask future songs so we don't attend to those during self attention. Our best Transformer models for both problems use 2 Encoder/Decoder Layers, 2 Multi Head Attentions and an input and embedding and attention dimension of 256 with 0.2 dropout.

2.2.2 BERT Encoder

In addition we try to extend other Transformer approaches like Bert4Rec[11] to see if they can better model these problems. Like the methodology in [11] we first pretrain a BERT Masked Language Model from scratch where the words are the song track ids. We utilize the Cloze prediction task by masking 20% of input songs randomly and then utilize the Transformer architecture to predict the masked songs. After pretraining we utilize this pretrained BERT-LM for fine tuning on the track sequence recommendation task where we only mask the tokens of the words in the second half of the session sequence which need to be predicted. For skip prediction we replace the 103K output layer with a binary skip output layer. Our best BERT model uses 4 Encoder/Decoder Layers, 4 Multi Head Attentions, input embed of 256 size, 0.2 dropout.

2.2.3 BERT Augmented Transformer

We also try modifying the Standard Transformer architecture by utilizing embeddings from 2.2.2 as input to the attention mechanism of a Transformer utilizing the architecture proposed in the paper *Incorporating BERT into Neural Machine Translation* [15] (Figure-2). In the Encoder we modify the self attention phase, by using BERT embeddings as a query and the input from the previous attention layer as the key and value into the attention scheme. We do the same for the Decoder. Additionally we use the dropnet trick to add regularization also described in [15] where we randomly with some fixed probability either only use Bert Encoded attention output, or the original self attention

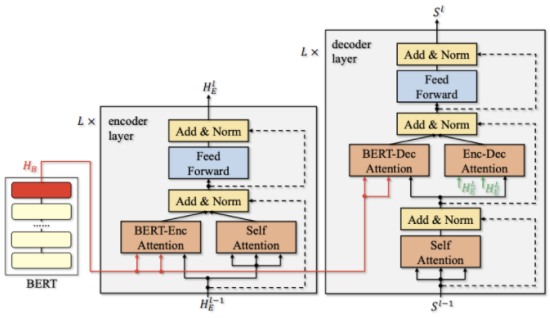


Figure 2: Bert Augmented Transformer Architecture. Credit: I*ncorporating BERT into Neural Machine Translation , Zhu et al*[15]

output or a weighted combination of both as input into the FFN of the Transformer. We use 2 Encoder/Decoder Layers, 2 Multi Head Attention Heads, input and embedding and attention dimension of 256 with 0.2 dropout.

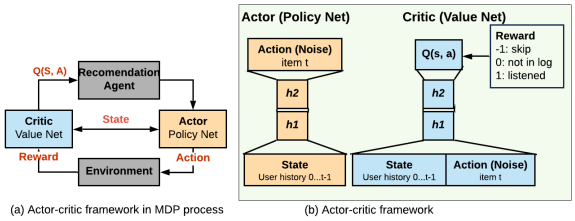
2.3. RL Based Architecture: DDPG Actor-Critic

Reinforcement Learning has it's advantage in modeling delayed reward in long sequences. Considering the limitation on discrete action space in DQN [10], we chose the actor-critic based method Deep Deterministic Policy Gradient (DDPG) [9] to be able to learn action embedding for high dimension (103K) track item space. (Figure-3). The recommendation problem was modeled as a Markov Decision Process, which consists of a tuple of (S, A, R) as follows: State s: user past browsing behavior up to time t-1, Action A: user's next item to recommend at time t, and Reward R: for skip prediction -1: skip, 0: not viewed, 1: listened, for sequence recommendation 1: match, 0: not match.The DDPG network is composed of two networks:

Actor network (Policy Net): Given a state we get action embedding, compute dot product between state embedding and action embedding of all items in action space, then choose the action (track) that best matches state embedding.

Critic network (Value Net): Critic network takes the action predicted from the actor network and concatenates with the state embedding to predict the $Q(s_t, a_t)$ and evaluates the reward observed.

The actor and critic network share the same state and action embedding, so that the critic feedback can modify the action taken in the next step from the actor (Figure-3b). In this task, we used track embedding size of 16 and hidden size 8 for both actor and critic network with batch size 512. This actor-critic framework was implemented according to the recent work from [14] (Figure-3), but with a few custom modifications to fit our data: 1) The state is concatenated from session embedding, track embedding and the product of the two to dynamically encode user's past interaction with tracks like in the classical wide and deep [2] recommendation structure. 2) We use negative sampling [3]



(a) Actor-critic framework in MDP process (b) Actor-critic framework

Figure 3: DDPG actor critic framework (Credit: Modified from [14])

with 1:99 positive:negative ratio to expedite convergence in large track space of 103K. 3) We add noise with decay in the actor network to add exploration to the original deterministic actions (See experiment3.1). In the original implementation in [14], the action selection from the policy net in the actor-critic network is deterministic via a dot product between action embedding and track embedding. To increase exploration at the early steps of the agent, we used a mask to randomly drop a fraction of dimensions in the dot product, while decaying the fraction as the agent gradually finished exploring the space. Unlike the epsilon-greedy methods that take a random action in exploration, this mask allows the actor to take a semi-educated guess of the next optimal action with some randomness. This is especially important for large item space exploration.

2.4. Training and Evaluation

For the NLP based approaches we train using Cross Entropy Loss as loss function for both problems. We train on a subset of 3.3M sessions and use a separate 100K sessions log for Testing. We use Adam Optimizer with default betas 0.99 and 0.999, batch size 256, and learning rate 1E-4 for Track Sequence Recommendation , and learning rate 1E-5 for Skip Prediction problem. We find these parameters after some grid searching and notice that smaller batch sizes and bigger learning rates led to poorer performance. We do not use any special scheduling rate and train until max 5 epochs. For evaluation we use Mean Average Accuracy (MAA) as our evaluation metric defined as follows.

$$MAA = \frac{\sum_{i=1}^{T} A_i * L_i}{T} \quad (1)$$

where T is the number of tracks to be predicted in a given session, A_i is the accuracy up to position i of the sequence, L_i is the boolean indicator for if the i'th prediction was correct. The benefit of using MAA is that like its more standard cousin Mean Average Precision (MAP) it weights accuracy by location in a sequence. If a song is correctly predicted earlier in the sequence it will boost MAA more than if it was predicted later in a sequence something which can be important from a user engagement perspective.

Method	Train MAA	Test MAA	Train Loss	Test Loss
Baseline(Predict All Skips)	N/A	30.5	N/A	0.328
Standard-LSTM (No Skip Padding)	52.0	52.1	0.563	0.562
Standard-LSTM (Skip Padding)	**60.5**	**61.1**	**0.527**	**0.526**
LSTM-Word2Vec	61.2	61.4	0.525	0.523
Bidirectional GRU	61.3	61.3	0.520	0.523
Standard Transformer	60.5	57.1	0.603	0.625
BERT Augmented Transformer	64.5	44.1	0.664	0.714
DDPG without Noise	N/A*	43.0	N/A	N/A
DDPG with Noise	N/A*	43.7	N/A	N/A

Table 1: Experiment Results for Skip Prediction Problem

3. Experiments and Results

3.1. Skip (Engagement) Prediction Task

We now provide the results of our different modelling experiments first on the skip prediction task in Table 1 above. Additionally we show that a naive baseline would perform with 30.5% MAA if we predicted all songs as skipped.

Experiment 1: LSTM Skip Track Masking vs No Skip Masking. We observe that the Test MAA shows significant improvement from 52% to 61% for Standard-LSTM when we mask input skipped tracks to special SKIP token while encoding. We hypothesize this is because the skipped tracks in the second half of the session are similar to the skipped tracks in the first half and that by padding Skipped songs to their own unique token we get some information about a user's previous skipping behavior. Since input skip padding performs so well we utilize this for all remaining NLP experiments mentioned below as well.

Experiment 2: Initializing Track Embeddings with Pre-trained Word2Vec Embeddings. Adding word2vec embeddings hardly show any increase in test MAA vs LSTM without Word2vec with 61% MAA each. This highlights the inherent bias in the data and the model. We observed the same trend when we tried using word2vec embeddings in Transformer based models too.

Experiment 3: Using Bidirectionality + GRU cells Using our standard LSTM model we compare its performance using a Bidirectional GRU Encoder-Decoder and we notice no noticeable improvement in performance.

Experiment 4: Using Transformer Architecture in place of LSTM We see that adding Transformer architecture to the skip prediction task actually worsens MAA with only 57% MAA with some more overfitting observed.

Experiment 5: Augmenting Transformer Architec-

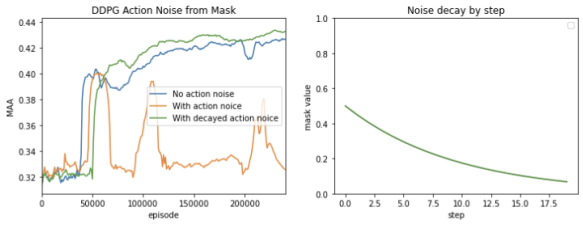


Figure 4: DDPG exploration with actor noise

ture with BERT Embeddings. Next we try the BERT augmented Transformer model described earlier in 2.2.3. We see performance only worsen with 44% Test MAA and overfitting to get severe (Train MAA = 66.4%) when using BERT embeddings as augmenting input into Transformer.

Experiment 6: DDPG -Adding Exploration noise in deterministic policies using mask decay . Here we try to replace the deterministic action from policy net in DDPG model [14] with an explore-exploit mechanism to reduce over-fitting on early steps of the agent. As shown in figure-4, the MAA metric with action noise alone is very unstable, since the actor is learning and forgetting by constantly dropping the learned policy from the mask. We then added a decay to the mask value, to gradually reduce the proportion of exploration while the agent is making further steps. The MAA curve for the actor with decayed noise is smoother compared to the original actor without noise, because the exploration at the early steps avoided the agent to be trapped at local optimum and converged too fast. Meanwhile, the agent takes more episodes to converge as a result of the exploration-exploitation trade off. While it learns and converges slower, at the end, the model avoided some local optimum and achieved a higher global MAA on test set.

3.1.1 Discussion

Overall we conclude that the best performing model, the unidirectional LSTM based Encoder Decoder, is also the simplest with 61% Test MAA. It is surprising to note that the Transformer model actually performs slightly worse with only about 57% Test MAA. One reason for this observation could be because our data defines input and predicted output session sizes to be only length 10. Since our input session sequences are still fairly small they may not suffer from the traditional problems of not capturing long term dependencies which might plague very long session sizes in LSTM models. So given the small session size the strengths of Transformers are not as apparent and theoretically less efficient approaches like LSTMS actually do better. This might also explain why adding Bidirectionality or GRU did not lead to improved performance as the small session size did not necessarily cause the model to forget earlier stages of an input. However if our session sizes were very big it is plausible that our results could have been different. And also RL methods might perform better because they can optimize for delayed rewards in a long sequence.

Method	Train MAA	Test MAA	Train Loss	Test Loss
Baseline (random guessing)	-	9e-11	-	-
LSTM + Word2Vec	9.78	0.80	7.59	10.39
Standard Transformer	**31.9**	**17.8**	**3.65**	**7.62**
BERT Encoder (Fine Tuned)	24.3	14.4	3.43	7.18
Bert Augmented Transformer	41.3	15.7	1.72	9.61
DDPG	38.5*	3.01*	-	-

Table 2: Results for Track Sequence Prediction Problem

3.2. Track Sequence Recommendation Task

For the track sequence recommendation problem we experiment using some of the same architectures as the Skip Prediction task with results shown in Table 2 above.

We see that this time the best model is the Standard Transformer easily outperforming the LSTM sequence based model with about 17.8 percent MAA Test Accuracy while the LSTM sequence model does a poor job learning sequences of tracks to predict with not even 1% MAA. We hypothesize that this is because our track sequence recommendation problem is very similar to the Language Modeling task where we model the probability of a sequence given a previous one. In such problems it is very important to give more importance to some words relative to others. LSTM/GRU based Seq2Seq may not do this as explicitly as Transformer methods which utilize self attention and multi head attention to better attend to all words and emphasize certain inputs in a sequence. In skip prediction, the exact ordering of tracks may not be as important as the actual skip behavior of a user (e.g: a user is bored and always skips songs) but in sequence recommendation the track ordering may be more important in determining the next sequence of songs that follows. Moreover, the track sequence prediction problem is more difficult as it involves correctly predicting one of 103K songs in the output (vs just a boolean output for skip/no skip). Since output softmax layer calculates scores over a larger 103K dim layer, it requires far more parameters and may need a more complex Transformer model.

But we also note that the more complex changes made to the Transformer models like augmenting with contextual embeddings of BERT into the model's attention mechanism do not lead to improved performance with only 15.7% Test MAA and actually worsen the overfitting observed as seen in the learning curves in Figure 5. This observation holds true even when evaluating accuracy at different positions in

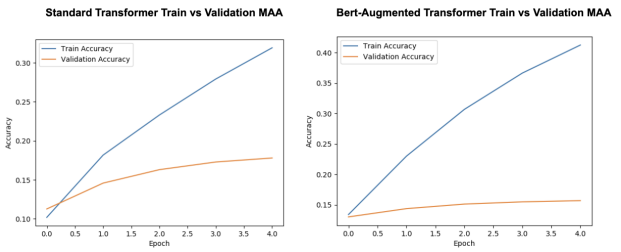


Figure 5: Sequence Prediction Learning Curves for Standard Transformer vs Bert Augmented Transformer

Model	T=1	T=3	T=5	T=7	T=9
LSTM Seq2Seq	0.75	0.60	0.61	0.60	0.55
Standard Transformer	27.7	21.7	18.3	15.6	13.1
Bert-Augmented Transformer	24.6	19.5	16.5	14.0	11.8

Table 3: Test Accuracy % at Position T in Predicted Sequence

the sequence with Bert Augmented Transformer underperforming at all positions in sequence. (Table 3).

This might be because BERT embeddings while great for Natural Language may not be as effective for predicting user-item sequences. BERT's original strength was that it was able to learn different "contextual" semantics of words but our context of songs may not be as multi contextual as natural language itself. For example a song may be associated with a genre or time period but it is less likely to have different "meanings" depending on the track sequence it occurs in vs a word like Apple which might refer to the fruit apple or the company Apple depending on context. As such we may not see improvements by integrating BERT based ideas for track sequence generation because the recommendation task we are solving may inherently not have the same contextual ambiguities as natural language. As such it may only over-parameterize the model causing it to overfit more.

One other reason why Transformer does so well can be observed from what the model is learning. Seen from Table 4 is the number of tracks correctly predicted in the sequence on hidden Test data with the values indicating the percent of total sessions with such number of correctly predicted tracks. We see that there is a huge spike in percentage of sessions with perfect 10/10 correctly predicted sequences with 11.6% sessions predicted perfectly for the Standard Transformer and only 9.6% for the less optimal Bert Augmented Transformer. We hypothesize that the model is successfully learning common sequences which are actually "track playlists" thereby boosting performance. These are either user or Spotify curated playlists that many users may be listening to. However, while it is good that the Transformer model is able to learn these common "playlist" sequences they also tend to change over time (e.g: Global Top 100) and the model's over reliance on such playlist sequences may cause overfitting on new data and might explain why even the best Transformer has big overfitting between Train (31.9%) and Test MAA (17.8%).

	0	2	4	6	8	10
Std Transformer	66.5	3.4	1.6	1.4	1.2	**11.1**
Bert-Augmented	67.8	3.6	1.6	1.2	1.1	**9.6**

Table 4: Percent of Total Sessions vs Number of correctly predicted songs in sequence

We also see that DDPG method suffers from overfitting the most. This might be because the reinforcement agent iterates one session at a time. Since our session lengths are pretty short, the model might be overfitted to predict one single user's behavior in the training samples, as opposed to representing the universal user-item relationship.

4. Challenges

One challenge we faced was the large size of the track vocabulary which originally was 3.7M unique tracks. Training on such a large track vocabulary embedding or calculating the output softmax proved challenging both computationally and for learning. We had no choice but to reduce to only 103K songs. Even then both the state and action space were so big that RL methods especially struggled to learn. Hence, instead of using Cross-Entropy Loss we could have modified our loss function more suited to a recommendation ranking task and use negative sampling as Hidasi et al [6] which might have alleviated the large vocabulary space problem. To make training times feasible we were forced to use only 10% (3.3M sessions) of data but methods like BERT assume a truly vast corpus for best results.

5. Conclusion

We extended the simple next click recommendation prediction problem to a more complex variant of predicting the next sequence of user actions (and also predicting if a user engages with the sequence or not). We conclude that our 7 sequential DL models do show some success in modeling these problems with Transformers performing best for track sequence recommendation but simpler models like LSTM Seq2Seq doing best for skip (engagement) prediction. However some potential areas of improvement remain. We only considered sequential approaches but could also have compared to other more common DL recommender approaches that use Factorization (DeepFM) [4] or Neural Collaborative Filtering [5]. We also considered sequences of length 10 but it remains debatable how effective such methods might be for very long sequences. Finally for sequence decoding we considered Greedy decoding but Beam Search could show improved performance. If given more time, these are areas we would have liked to explore.

References

[1] Brian Brost, Rishabh Mehrotra, and Tristan Jehan. The music streaming sessions dataset. In *Proceedings of the 2019 Web Conference*. ACM, 2019.

[2] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[3] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[4] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for ctr prediction, 2017.

[5] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering, 2017.

[6] Balázs Hidasi and Alexandros Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, Oct 2018.

[7] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. 2016.

[8] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. 2018.

[9] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[11] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1441–1450, 2019.

[12] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[14] Xiangyu Zhao, Liang Zhang, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. Deep reinforcement learning for list-wise recommendations. *arXiv preprint arXiv:1801.00209*, 2017.

[15] Jinhua Zhu, Yingce Xia, Lijun Wu, Di He, Tao Qin, Wengang Zhou, Houqiang Li, and Tie-Yan Liu. Incorporating bert into neural machine translation, 2020.

6. Work Division

The team contributions can be found in Table 5.

7. Code Documentation

We used Pytorch for our modeling. All our model code and training scripts can be found on:
`https://github.com/cuitianyuan/sequential_recommendation`. You can follow the README.md to run the code.

7.1. Implementation Notes

1. For Transformer based Architectures we used the Pytorch nn module to define both the Transformer and BERT models.

2. For the customized BERT Augmented Transformer we modified the Pytorch nn module source code to implement the custom attention layers and dropnet trick described in the paper [15]. You can view our modified architectures based on Torch source code in our repo under models/CustomizedTransformerDropNet.py

3. Code for DDPG [14] offline reward simulation was used as starting point, Actor-Critc network was modified as detailed in 2.3.

4. We used and modified this for LSTM based architecture.

Student Name	Contributed Aspects	Details
Ahmad Khan	Implementation of the 3 Transformer Based Modeling Architectures and their Analysis, Data Processing Script and Evaluation Script	All coding Implementation of the the 3 Transformer Based Models including Standard Transformer. [13], BERT pretraining and finetuning based on Bert4Rec paper [11] , and the Bert Augmeneted Transformer [15]. Plus helping debug LSTM track sequence model. Model Analysis of Transformer based Models. Wrote the Data Processing Script used by all team members to parse and generate the training and test data. Coded and provided Evaluation setup including Mean Average Accuracy implementation.
Tianyuan Cui	Implemented and improved DDPG actor-critic network for reinforcement learning based on [14]	Improved the architecture from the paper by adding wide and deep state embedding, negative sampling for large item space and actor noise with decay to balance exploration-exploitation to reduce overfitting (Fig-3).
Sagar Arora	Implementation and Analysis of LSTM based architectures	Implementation of 4 different LSTM based architecture variants (standard lstm without padding, standard lstm with padding, lstm-word2vec and bidirectional gru) for the skip and sequence prediction problems. Pretrained word2vec embeddings for tracks were also tried by team members.

Table 5: Contributions of team members.