Content-Based LSTM Network for Estimating Music Play Counts

KYLE SALITRIK, The Pennsylvania State University, USA
TOMOKI TAKASAWA, The Pennsylvania State University, USA

In the current state of the music industry, about 1% of artists make up nearly 80% of the total revenue in the market. The aim of our research is to create an assistant for recommendation systems that estimates the number of times a song would be played by a particular user, and therefore generate the most revenue for artists. By utilizing only audio data, we construct a system that is ignorant of an artist's popularity or other metadata such as genre. It provides a win-win-win situation for musicians, consumers, and record labels: Users will be able to discover new artists, up-and-coming artists could be recommended more often and allow them to gain a larger fanbase, and record labels will generate more revenue due to these increased plays.

CCS Concepts: • **Computing methodologies** → **Neural networks**;

Additional Key Words and Phrases: Music Recommendation, LSTM

1 INTRODUCTION

In recent years, the influence of the music recommendation system on the recording industry has become much more significant than it was before due to the emergence of online music streaming services. The current recommendation systems utilize a collaborative filtering approach on the metadata of music to characterize songs. While this method is a very intuitive way to estimate users' preference, it raises several problems for users and music artists if the recommendation system solely relies on metadata. Because songs are characterized by metadata such as artists, the songs by popular artists are much more likely to be recommended to users. This is problematic for mainly two reasons.

First, it limits the user's ability to discover new artists and songs. The purpose of browsing songs is generally to discover more likable songs beyond artists or genres. If the recommender system keeps suggesting songs that have already been liked or artists the user follows, it may conflict with the user's interests. Furthermore, users may never get an opportunity to cultivate the interests towards other types of artists or songs in different genres if only similar songs are being suggested. Second, it creates an unfair disadvantage for unknown artists. The top 1% of artists account for the vast majority of all revenue from recording music. [1] While it is true that the top artists are more prolific than majority of artists, the music recommendation system that solely based on the content of music can raise the visibility of undiscovered, talented music artists.

Several recent studies addresses such a problem, but more research can be done on the subject. Hence, the goal of our research is to provide an additional metric that can improve the quality of music recommendation systems. In order to eliminate the problem discussed above, this paper proposes a method of analyzing music and users' preference that is solely based on raw music data. Furthermore, we propose an alternative approach to understand users' preference of songs. Traditional music recommender generally only predicts whether or not users would like the song. However, such a metric cannot tell how much users would enjoy the song. In this paper, we utilized Long Short-Term Memory regression model to estimate the music play counts by users. The music play counts are often proportional to how much user enjoys a song, and it is useful to predict songs that users are likely to enjoy the most. In addition, such a recommender feature could be potential application for industrial use, since the interests of distribution companies and music artists is how many times listeners would play their songs.

2 PRIOR RESEARCH

2.1 Prior Music Recommendation

2.1.1 Deep Content-Based Music Recommendation [13]. The Deep Content-Based Music Recommendation

Authors' addresses: Kyle Salitrik, The Pennsylvania State University, State College, PA, 16801, USA; Tomoki Takasawa, The Pennsylvania State University, State College, PA, 16801, USA.

Publication date: April 2018.

paper proposed a very unique approach to address problems with the current music recommender systems. In this paper, they have attempted to analyze a latent factor vectors, which is a compact description of userś tastes and corresponding characteristics of items, based on the preference in past. Because the music audio signal is used to characterized music, this approach is less likely to result in an unfair result for unpopular artists. They have tested two approaches to predict latent factors, a method using Bag of Words representation on Mel-frequency Cepstrum Coefficients and a method using Convolutional Neural Networks.

2.1.2 Improving Content-based and Hybrid Music Recommendation Using Deep Learning [15]. Music recommender system with traditional content based music analysis, which uses high level property of music such as Mel-frequency Cepstral Coefficients, often has unsatisfactory result in accuracy. This paper utilized a novel model and probabilistic graphical model to recommend songs by using both collaborative filtering of metadata as well as analysis on learned feature from audio content. Their Hierarchical linear model with a deep belief network particularly improves the accuracy of prediction in a warm start stage.

2.1.3 Collaborative Deep Learning for Recommender Systems [14]. In this paper, a method called Collaborative Deep Learning was proposed to address the problem with information insufficiency. With music recommendation systems using Collaborative Filtering, the accuracy drops significantly when the ratings given by users are insufficient. On the other hand, when recommending music using latent representations, the result may not be very effective if the auxiliary information is insufficient. While a statistical model, Collaborative topic regression, can be used to bridge collaborative filtering methods and latent representation methods, lack of sufficient information can still degrade the accuracy of recommendations. However, this new method, collaborative deep learning, solves this problem by jointly performing deep learning for the content information and collaborative filtering for the ratings by using hierarchical Bayesian models.

2.2 LSTM [10]

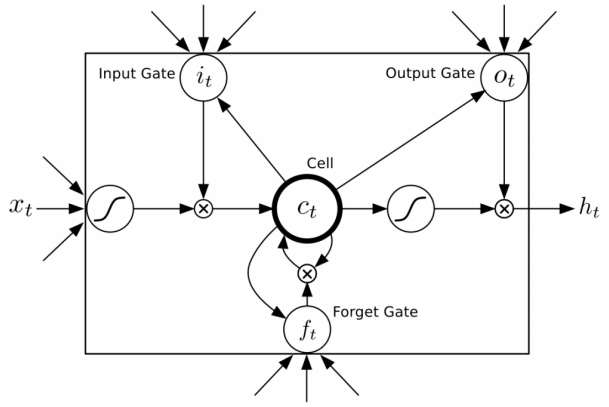

Fig. 1. LSTM Cell Representation

LSTM is a type of Recurrent Neural Network, specializing to work with long term dependencies. Typically, an LSTM network is composed of cells (depicted in figure 1) containing an input gate, output gate, forget gate, and at least one memory cell. The input gate controls what value can be entered into the memory cell, the output gate controls which of the values in the cell should be used, and forget gate controls which value in the cell should remain. By having such an architecture, a neural network utilizing the LSTM framework excels at understanding input information using what it has learned before, making it an ideal framework for audio analysis. The recent success in using Neural Networks for tasks such as speech recognition is due to this framework.

3 DATASET DESCRIPTION AND ANALYSIS

The discussion in this section covers the datasets that were used for obtaining the training data, how the training data used in our research was selected from these datasets, and the preprocessing methods used before feeding the data into the neural network.

3.1 DATASETS USED

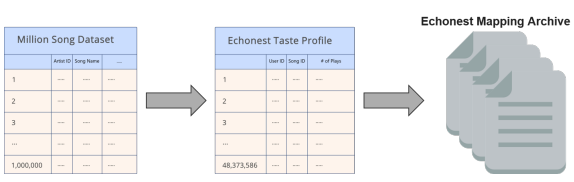

Fig. 2. Simple Graphic Displaying Datasets

3.1.1 Million Song Dataset. The Million Song Dataset (depicted on the left in Figure 2) is a dataset containing various fields of metadata for popular songs. [7] The data includes information such as the song title, artist name, album title, as well as more in depth information like the tempo of the song (BPM) and the estimated musical key that the song is played in. Many other researchers have used this dataset and it's subsets to perform music recommendation related research. [13–15] Unfortunately, all of the data provided by the subset is unusable for the purposes of the experiments performed here as no real-world user data is provided. However The EchoNest Taste Profile Subset provides this information and will be discussed next.

3.1.2 The EchoNest Taste Profile Subset. The EchoNest Taste Profile Subset (depicted int the center of Figure 2) provides play count information using 384,546 songs of the Million Song Dataset for 1,019,318 prior EchoNest users. [9] There is a total of 48,373,586 records that contain the EchoNest song ID, the user ID of the user playing that song and the number of times that user played a song. This information is immensely useful for researchers that wish to perform a music recommendation analysis, or in our case, estimate how many times a user is likely to play a given song.

With the limited ability of real-world user data that is readily available via APIs due to permissions needed to access a user's information on popular music services, this archive is very valuable for research. For example, Spotify's API requires explicit permission for a user to simply read a user's recently played and top played tracks. [2] Even then, these API queries are limited to 50 songs maximum, and are not guaranteed to not contain duplicates. [3, 4]

3.1.3 Echonest Mapping Archive. Due to the acquisition of The EchoNest by Spotify and it's subsequent shutdown, it is no longer possible to easily map the songs available in The EchoNest Taste Profile Subset or Million Song Dataset to other music services as this functionality was removed. However, a group of researchers created the The EchoNest Mapping Archive (depicted on the right in Figure 2) that maps the songs includen in the Million Song Dataset to many other music services such such as Spotify, 7-Digital, MusiXMatch, and others in an archive of individual JSON files. [8] Without this dataset, the only way to map between services would be by querying using Artist and Song names which is not guaranteed to provide accurate results. It is worth noting that the archive is not 100% complete. In our experiments, we were able to obtain between 40% and 60% of the Spotify IDs for songs we obtained during data extraction.

3.2 TRAINING DATA EXTRACTION

The final set of data that was used for training and evaluating our network models was obtained primarily from The EchoNest Taste Profile Subset and secondarily from The EchoNest Mapping Archive. The triplets from The EchoNest Taste Profile Subset were loaded into a MySQL database and results were obtained after various queries narrowing down the dataset to an acceptable level based on alloted computation time and storage space. These queries are explained in the below subsections. Once the final records were obtained, The EchoNest Mapping Archive JSONs were queried to obtain the corresponding Spotify IDs, added to a Spotify Playlist using the Spotify API and music data was obtained using a Spotify Premium Membership.

3.2.1 Limiting Maximum Number of Plays. After initially examining the dataset, it appeared that there were quite a few outliers due to some play counts. Some records showed over 1000 plays for a single user/song pair, so it was decided to trim outliers from the dataset by limiting the maximum number of plays per record to 200 plays. In total this trimmed the entire dataset from approximately 48 million records down to 48,370,466.

3.2.2 Obtaining the Top 5 Users. From the trimmed dataset, the top 5 users were selected based on the total aggregate number of plays. These were obtained using an SQL query to sum the number of plays, grouped by User ID. The decision to choose the top 5 users was done in order to be able to provide a comparison of network structures for multiple users and to see whether or not a model exists that performs universally acceptably. The decision to only perform network training on the data for 5 users was due again to limited training time and storage budgets.

3.2.3 Choosing Training Data. The final data set used for training the networks was obtained by querying the database for all songs listened to by at least two of the top 5 users. This resulted in a total of 341 songs, of which only 150 had corresponding Spotify IDs in The EchoNest Mapping Archive JSON files. These Spotify IDs were

then added to a Spotify Playlist and the audio files were downloaded using a Spotify Premium Membership.

3.3 DATA PREPARATION

This subsection describes how the user and audio data were treated during the training process and the preprocessing methods used before feeding in the data to the network.

3.3.1 User Play Information. Of the 341 final songs selected, a CSV file was generated for each user containing the records for that User ID with each record consisting of the User ID, Song EchoNest ID, and number of plays. In order to be able to compare the performance per user directly, the column of user plays was standardized for each user before being used as training data by the network. The function in Algorithm 1 was implemented for loading and preprocessing the user data.

ALGORITHM 1: Load and Standardize User Data

Input: Location of User Data, Location of Music Data
Output: Tuples of (User ID, Song ID, Standardized Play Value)

```
userData = read user record CSV file
musicAvailable = list songs in music directory
forall song s in userData do
    if Song s is not in musicAvailable then
        Remove s from userData
    end
end
Standardize number-of-plays column in userData
return userData
```

3.3.2 Audio Data. In order to make the network compatible with songs of varying length as well as make recommendations based on partial song information, the input to the network was limited to 5 second song clips. As will be explained in Section 4, the number of these 5 second vectors consumed by the network varied based on the network architecture, however each individual clip was no longer than 5 seconds. In order to process the audio files, two domain-specific Python libraries were employed: PyDub and LibROSA. [5, 6] In this case, PyDub was used to split the songs into their 5 second increments (depicted in figure 3) and LibROSA was used to convert the split MP3 files into NumPy arrays.

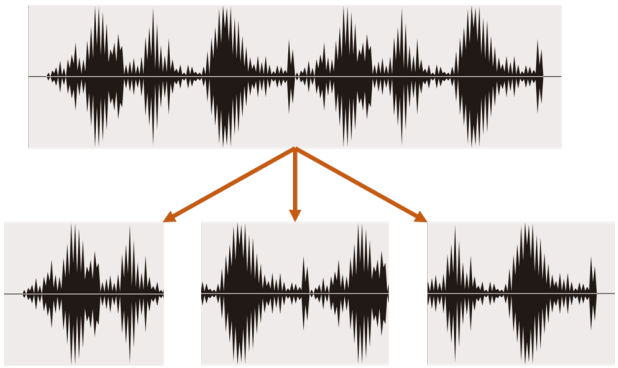

Fig. 3. Figure Depicting Audio Split

4 NEURAL NETWORK MODELS

4.1 Software and Hardware Used

For defining and training the models, Keras 2.1.4 was used with Tensorflow 1.7 on four AWS Elastic Cloud Compute (EC2) AMI instances. Keras was used in order to streamline the process of creating and defining the neural networks. The loaded music data was split into 90% training and 10% validation (the low ratio of validation to training data was due to having a small dataset) and each network was trained on 100 epochs with sample sets being shuffled in between each epoch.

The choice between one or two LSTM layers as well as the input tensor of the network was defined at runtime using parameters passed to the Python script. The following subsections will explain in detail the concept of preparing the input data as well as the exact definitions of 'data vector', 'time steps', 'training samples', and 'input tensor' for the purposes of this paper.

4.2 LSTM Input Structure

The following algorithm (2) walks through the pseudocode used to load the audio data into the accepted input tensor based on the parameters passed to the python script.

4.2.1 Data Vector. The phrases 'data vector' or 'clip' refer to the NumPy array of a 5 second audio segment of the song. For example, a 60 second song would be broken up into twelve 5 second clips or data vectors. The reason for this explicit explanation is due to the Keras LSTM input shape definition which uses the phrase 'time steps' to refer to a different concept. In the case that a song was not evenly divisible into 5 second clips, the remainder

ALGORITHM 2: Load Music and Target Tensors

Input: Pre-processed user data, music data location, number of time steps

Output: Tensor of dimensions: (training samples, time steps, data vector), Vector or matrix of target values

```
targets = empty list
inputTensor = empty list
forall songs s in userData do
    segmentList = sorted list of segments in song directory
    trainingSamplesForSong = floor(length(segmentList) / timeSteps)
    for i in range(0, trainingSamplesForSong) do
        append target to targets
        currTrainingSample = empty list
        currentClip = i * timeSteps
        for j in range(currentClip,
          currentClip + timeSteps) do
            load and append column of segment data to
              currTrainingSample
        end
        append currentTrainingSample to inputTensor in
          3rd dimension
    end
end
return inputTensor, targets
```

of the final clip was expanded to 5 seconds by adding silence to the remainder of the clip.

4.2.2 Time Steps. When calling the Python script to begin training the neural network, the second parameter passed determines the number of time steps that will be used in the network. In this case the phrase 'time step' refers to the number of 5 second clips that are passed as a single training sample to the network. For example, a network with three time steps in the input shape will take in three 5 second clips as a single training sample. In the experiments run, the number of time steps was varied between 1 and 24 as the longest song available was only about 2 minutes long.

4.2.3 Training Samples. The number of training samples fed into the network was determined by both the number of songs and the length of each individual song. Due to this, each training sample for a single song would be treated as an individual song, except in the case that the song was only long enough to create a single training sample. As an example, if a song was 15 seconds long and the number of time steps was set to 1, then that song would be broken up into 3 training samples, as shown in figure 4. As another example, if a particular

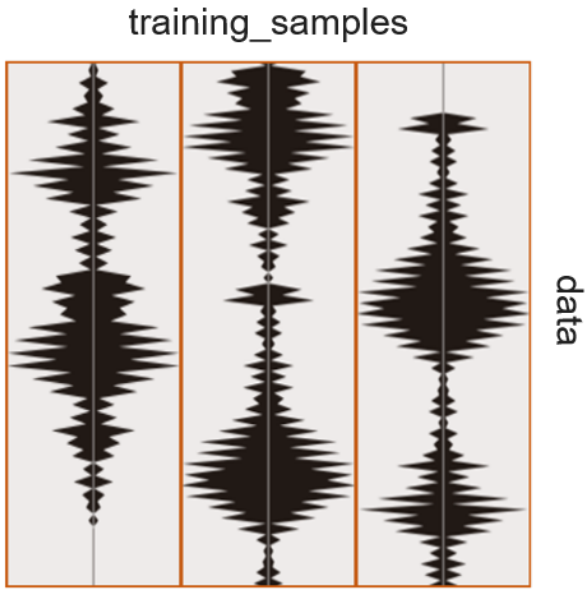

Fig. 4. Depiction of Input Using a Single Time Step

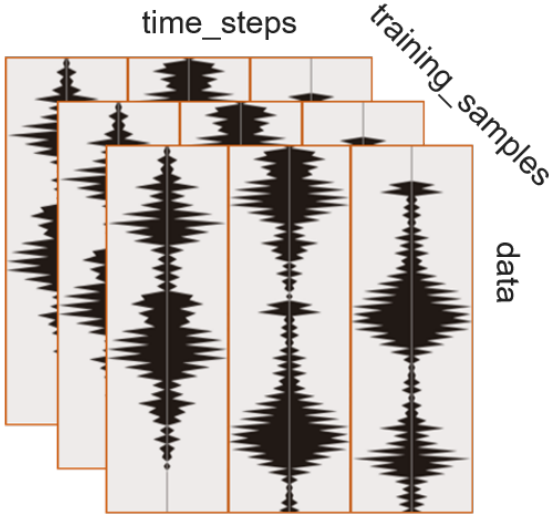

Fig. 5. Depiction of Input Using 3 Time Steps

song was 45 seconds long and the number of time steps was set to 3, then one would end up with 3 training samples for that song, each with three 5 second clips, as displayed by figure 5, and this song's target would be added 3 times into the targets vector, with each training sample being treated individually.

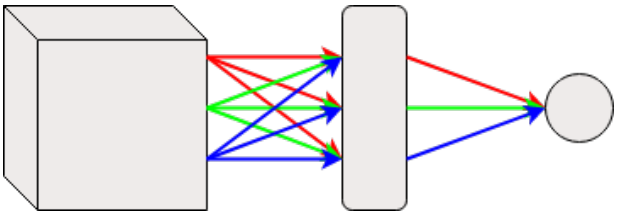

Fig. 6. 1-Layer LSTM Model

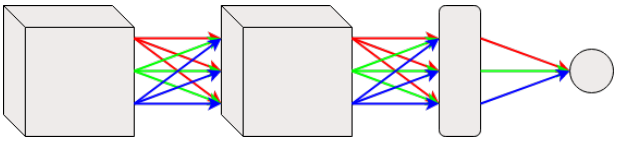

Fig. 7. 2-Layer LSTM Model

4.3 Models Used

For the experiments, as described at the beginning of the section, two different model shapes were used. First was a single LSTM layer with input shape defined as described previously, followed by a dense layer with a single output node depicted by figure 6. The second network architecture used was simply adding another LSTM layer in between the first layer and dense layer as shown in figure 7.

5 RESULTS

In this section, we discuss the results for each model type: single layer and two layer LSTM, the model that performed best 'universally' across almost every user, the best performing model per user, and the best performing models for a single user (User 3). User 3 was chosen not because they had the best performance out of all users, as User 2 had that distinction, but rather User 3 has a balance of significant improvement and performance. An examination of poor performance is also included as to be transparent and address criticisms of our approach. It is worth noting that for all of the following figures and discussions, the metric for the loss is the mean squared error (MSE) of the standardized play counts. In other words, an MSE of 1 means that the error by the network was ±1 standard deviation. Over all experiments, user 1 had the worst performance, with no final MSE being under 1 standard deviation.

5.1 Single Layer Model Results

Depicted in figure 8 is the single layer LSTM model that had the best improvement for each user. These models

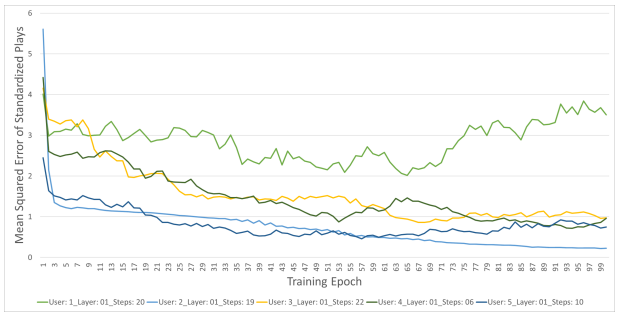

Fig. 8. Best Performing Single LSTM Layer Model Per User

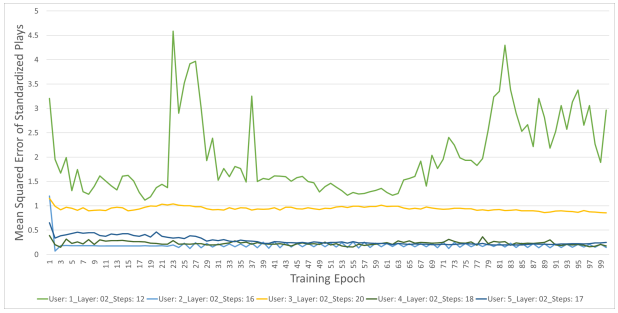

Fig. 9. Best Performing Two LSTM Layer Model Per User

are not necessarily the ones with the best performance overall. As can be seen, the number of time steps varied greatly, however in the majority of cases, the best improvements came from models with ≥ 10 time steps. In this case, excluding User 1, the MSE for each model was < 1 standard deviation.

5.2 Two Layer Model Results

Similar to the prior subsection, figure 9 displays the two layer models with the most improvement per user. As can be seen, in most cases the performance of the two layer LSTM models is significantly better than the single layer models. Excluding User 1, all of these models use ≥ 16 time steps, further indicating that a longer time history improves the performance of the network.

5.3 Universal Model

While no model performed best for every single user, the combination of two LSTM layers and 18 time steps was the closest model to achieving that goal and the performance of this model is plotted in figure 10. In this case User 1's performance was divergent, however all other users had a final MSE of < 1 and User 2-4 had a final MSE < 0.4 standard deviations. Once again, this

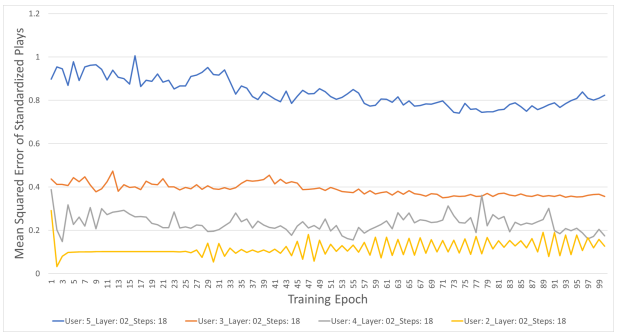

Fig. 10. Best Universally Performing Model: 2 Layers 18 Time Steps

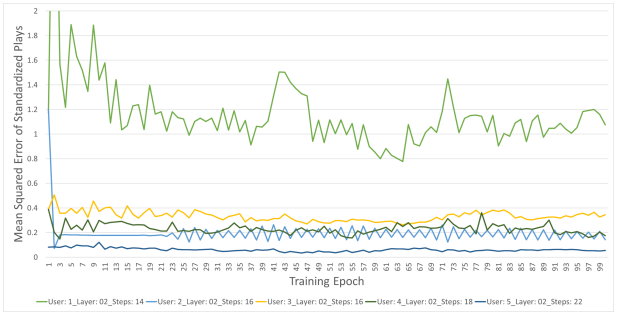

Fig. 11. Best Performing Model Per User

supports the assumption that a larger history for the LSTM to consider provides better performance.

5.4 Best Model Per User

Next we examine the best models in terms of performance per user and a sample of models for User 3 demonstrating excellent improvement and overall performance. Firstly, in figure 11, for Users 2-5, the final performance was MSE < 0.4, which is better than a random guess at the standardized number of plays for that user. User 5 had the best performance of all users, with the best performing model having a MSE < 0.06.

In figure 12, multiple combinations of models are displayed over all 100 training epochs. In this case, the best performing models were under 0.5 MSE, however all models converged to an MSE of < 1 standard deviation. The combination of 1 LSTM layer and 22 Time Steps is particularly impressive, converging to < 1 MSE from > 4.15 MSE at the first iteration.

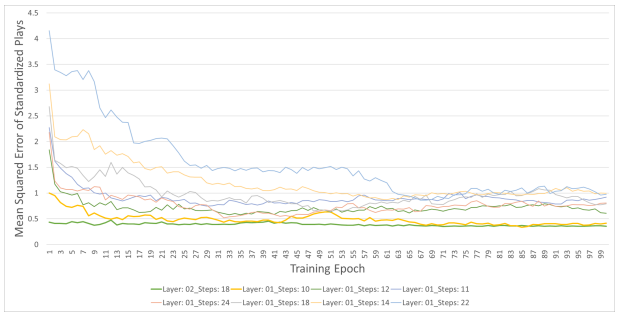

Fig. 12. Sample of Model Performance for User 3

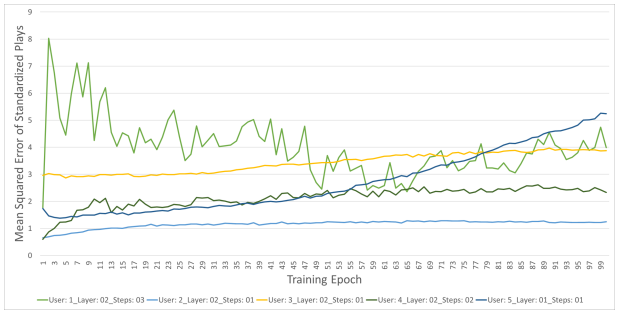

Fig. 13. Examples of Divergent Models Due to Too Few Time Steps

5.5 Criticisms

For all users, there were some model combinations that had diverging loss, most commonly when the number of time steps was low. The researchers assume that this is due to the nature and purpose of LSTM networks, a longer time step (history) creates better performance. After all, the goal is to learn a pattern over time and remember that pattern to make predictions. However, it is worth noting that this was not true 100% of the time. As one can see in figure 13, these low time-step models provide poor performance and a divergent mean squared error. Some models even with larger time step values also performed poorly, most likely due to a bad random initialization.

6 CONCLUSION

Most research in the field of music recommendation relies on binary classification or categorizing music into clusters. While there has been success in this area, many of these methods still rely heavily on metadata to accomplish their goal.[14, 15] Although the success of our experiments was relatively minor considering the constraints that time and budget imposed on the amount

of data that was able to be processed, the results were fairly promising and indicate that further research can be fruitful.

As stated, our goal was to find a method to assist current music recommendation systems while being metadata agnostic. Agnosticism of information such as the musician(s) responsible for writing a song creates potential for new artists to be recommended equally as well established ones, in essence creating a level playing field in the market and a lower barrier to entry. The content-based nature of the neural network creates an environment in which latent features may be extracted regardless of genre or artist and theoretically allow users to expand the repertoire of music that they enjoy. In addition, the focus on predicting the number of times a user will play a song potentially allows the distribution service, record label, and music artists to increase revenue.

7 FUTURE WORK

The following suggestions for improvement of the developed system could prove to be fruitful paths to take.

- Attempting to use a deeper or varied network:
  - Adding more LSTM Layers
  - Adding more LSTM layers with the number of hidden neurons cascading
  - Using more fully connected layers with a slower reduction to one output neuron
  - Allowing dropout internally in the LSTM layers or in the dense layers
  - Using an autoencoder to preprocess audio data before feeding into the LSTM network.
- Increasing the amount of data for training the networks
- Using different loss functions than MSE
- Using different activation functions other than sigmoid in the neural network
- Use only a single time step with multiple networks and averaging the result of all networks

APPENDIX A

All source code used for the project can be obtained from GitHub - ttakasawa: Music Play Count Estimator

REFERENCES

[1] 2014. The Top 1Income, Study FindsâĂę . https://www.digitalmusicnews.com/2014/03/05/toponepercent/

[2] 2018. Authorization Scopes | Spotify for Developers. https://beta.developer.spotify.com/documentation/general/guides/scopes/

[3] 2018. Get a User's Top Artists and Tracks | Spotify for Developers. https://beta.developer.spotify.com/documentation/web-api/reference/personalization/get-users-top-artists-and-tracks/

[4] 2018. Get Current User's Recently Played Tracks | Spotify for Developers. https://beta.developer.spotify.com/documentation/web-api/reference/player/get-recently-played/

[5] 2018. jiaaro/pydub @ GitHub. http://pydub.com/

[6] 2018. LibROSA âĂŤ librosa 0.6.0 documentation. http://librosa.github.io/librosa/

[7] 2018. Million Song Dataset. https://labrosa.ee.columbia.edu/millionsong/

[8] 2018. Million Song Dataset Echo Nest Mapping Archive. https://labs.acousticbrainz.org/million-song-dataset-echonest-archive/

[9] 2018. The Echo Nest Taste Profile Subset. https://labrosa.ee.columbia.edu/millionsong/tasteprofile

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

[11] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. 2015. Unsupervised Learning of Video Representations using LSTMs. (2015). https://doi.org/citeulike-article-id:13519737 arXiv:1502.04681

[12] Martin Sundermeyer, Hermann Ney, and Ralf Schluter. 2015. From feedforward to recurrent LSTM neural networks for language modeling. *IEEE Transactions on Audio, Speech and Language Processing* 23, 3 (2015), 517–529. https://doi.org/10.1109/TASLP.2015.2400218

[13] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems 26*, C J C Burges, L Bottou, M Welling, Z Ghahramani, and K Q Weinberger (Eds.). Curran Associates, Inc., 2643–2651. http://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf

[14] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2014. Collaborative Deep Learning for Recommender Systems. (2014). https://doi.org/10.1145/2783258.2783273 arXiv:1409.2944

[15] Xinxi Wang and Ye Wang. 2014. Improving Content-based and Hybrid Music Recommendation Using Deep Learning. In *Proceedings of the 22Nd ACM International Conference on Multimedia (MM '14)*. ACM, New York, NY, USA, 627–636. https://doi.org/10.1145/2647868.2654940