DeepPlaylist: Using Recurrent Neural Networks to Predict Song Similarity

Anusha Balakrishnan
Stanford University
anusha@cs.stanford.edu

Kalpit Dixit
Stanford University
kalpit@stanford.edu

Abstract

The abstract paragraph should be indented 1/2 inch (3 picas) on both left and right-hand margins. Use 10 point type, with a vertical spacing of 11 points. The word **Abstract** must be centered, bold, and in point size 12. Two line spaces precede the abstract. The abstract must be limited to one paragraph.

1 Introduction

Modern song recommendation systems often make use of information retrieval methods, leveraging user listening patterns to make better recommendations. However, such systems are often greatly affected by biases in large-scale listening patterns, and might fail to recommend tracks that are less widely known (and therefore have sparser user listening history). In general, the field of music information retrieval calls for a deeper understanding of music itself, and of the core factors that make songs similar to each other. Intuitively, the audio content of a song and its lyrics are the strongest of these factors, and the ideal recommendation system would be able to use them to make conclusions about the similarity of one song to another.

Recent developments in deep neural network and recurrent neural network architectures provide a means to train end-to-end models that can predict whether two songs are similar based on their content. In this paper, we present two different end-to-end systems that are trained to predict whether two songs are similar or not, based on either their lyrics (textual content) or sound (audio content).

2 Related Work

Not much work has been done in the field of Music Recommendation. *Collaborative Filtering* has been the method of choice for music and video recommendation engines. approaches to Music Classification. In fact, Spotify already uses it to recommend millions of songs daily spo [2013] spo. Netflix suggests movies based on *Collaborative Filtering* Zhou et al. [2008]. But, in general *Collaborative Filtering* suffers from two major problems. Since it analyzes patterns in usage data, it needs a large amount of human usage data which is not always easily/cheaply available. Secondly, *Collaborative Filtering* has a Capitalistic nature. The popular songs get recommended more often, making them even more popular. This in turn dooms the fate of a new song which has a niche audience or a song from a new artist. Even more fundamentally, similar patterns in music listening habits is only an indirect indication of song similarity. It does not fundamentally measure the similarity between songs and hence has an upper bound on its performance.

An approach that fundamentally measure song similarity would address all the issues with *Collaborative Filtering*. Kong, Feng and Li Kong et al. used precomputed convolutional filters followed by an MLP to get 73% genre classification accuracy on the Tzanetakis1 dataset. Sander Dieleman used CNNs on raw audio vs spectrogram input in Dieleman and Schrauwen [2014] and found the spectrogram input to give better results. In Dieleman and Schrauwen [2013], Sander Dieleman also used CNNs on raw audio data to predict song embeddings produced by standard (and powerful) collaborative filtering methods.

3 Approach

3.1 Problem Statement

Our general problem of similarity prediction can be modeled as a pairwise sequence classification problem as follows:

Let the two songs P and Q be represented as a sequence of vectors $(P_1, P_2, P_3, ...P_n)$ and $(Q_1, Q_2, Q_3, ...Q_m)$, and let $Y \in 0, 1$ be an indicator representing whether P and Q are similar songs or not. For our specific problem, each vector P_i can be either the word vector x_i of the i^{th} word in the lyrics of song P, or the audio spectrogram for a fixed number of frames (audio samples), representing the audio spectrogram for the t^{th} "chunk" of time in song Q (see section 4.1).

Thus, our classification problem can be defined as

$$\hat{y} = f(P, Q)$$

, where $f(P, Q)$ is a highly non-linear classification function learned by the neural network described later in this section. We note that since the function f predicts the similarity between two songs, it must necessarily be symmetric in nature (i.e. $f(P, Q) = f(Q, P) = \hat{y}$.

3.2 Models

Both the architectures presented in this paper build upon a general Long-Short Term Memory (LSTM) framework. The motivation behind using an LSTM-based architecture stems from the fact that audio is inherently sequential in nature, and the similarity between two songs (particularly between their audio signals) must in at least some way be determined by the similarities between their sequences over time. While all recurrent neural networks (RNNs) serve the general purpose of modeling patterns in sequential data, LSTMs are often able to "remember" longer patterns and model them better than vanilla recurrent neural networks ([Hochreiter and Schmidhuber, 1997]). Since both the lyrics and audio of a song are often fairly long sequences of data, we chose to use an LSTM (rather than a conventional RNN) as the basis for both architectures.
General framework. Both songs P and Q are inputs to our model, and the output is a single classification label $\hat{y}$. In addition to LSTM units (recurrent units that contain a hidden state that is updated at each timestep), we experimented with adding several layers to our model, depending on the type of input (text or audio). In general, the input for each song (either the word vectors or the audio spectrogram) are unrolled one timestep at a time through the network, and the LSTM hidden state is updated at each timestep. At the final timestep t_{final}, the hidden state of the LSTM after a single forward pass of P is h^P_{final}, and the hidden state after a forward pass of Q is h^Q_{final}. These hidden states are then combined, and the resulting tensor is passed through one or more fully connected layers and a final softmax layer to produce a single binary output. We describe our experiments with different layers, combinations of hidden states, etc. in Section 4 below.
Convolutional layers. Time-based convolutions over audio spectrogram data have proved extremely effective in training end-to-end models for speech recognition and other audio-based tasks [Amodei et al., 2015]. Thus, for our audio-based classifier, we use one or more time-based convolutional layers to perform convolutions over the spectrogram data before it is passed to the LSTM as input.
Figures **??** and 1 show a generic view of the architectures we propose in this report.

4 Experiments

4.1 Dataset

The Million Song Dataset (MSD) [Bertin-Mahieux et al., 2011] is a huge repository that contains audio features, metadata, and unique track IDs for a million pop music tracks. One huge advantage of this dataset is that the track IDs in the dataset provide unique links to several other interesting

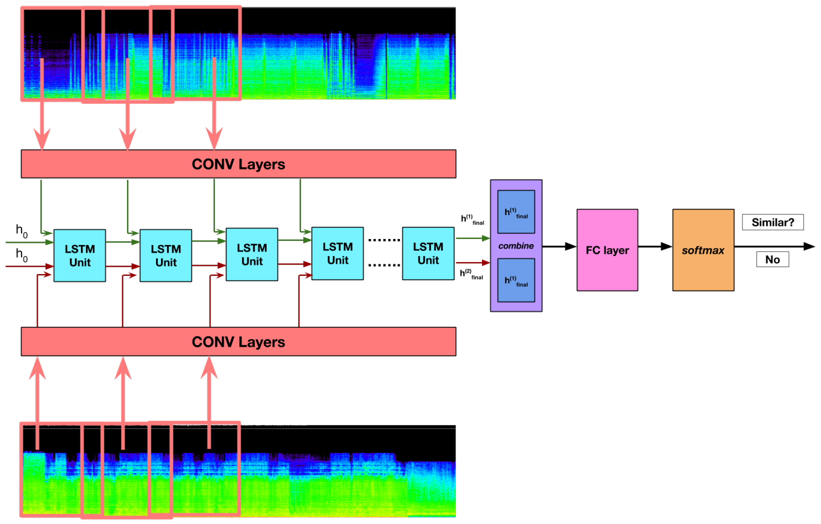


Figure 1: Our proposed model that performs similarity classifications based on audio spectrogram input. Each spectrogram is split into "blocks" of time, and 1D convolutions are performed across these time blocks. The outputs of the convolutional layers are passed to LSTM units.

sources of data. One of the largest datasets that the MSD links to is the LastFM dataset[1]. This dataset contains lists of similar songs for over 500,000 tracks in the MSD, and provides a similarity score for each pair of songs. We use this dataset to obtain the ground-truth labels for our classification problem. Each pair of tracks has a similarity score between 0.0 and 1.0, and we threshold this score at 0.5, such that any pair with a similarity score greater than or equal to 0.5 are considered similar, while pairs with lower similarity scores are considered "not similar". The MSD also makes it possible to query several APIs for additional information. Among these are the musixmatch API[2], which provides access to lyrics snippets, and the 7digital API [3],which provides audio samples for a large number of tracks contained in the MSD. We used these APIs to collect the lyrics and audio samples that our models are trained on.
We collected the lyrics for a total of 54412 songs, and audio samples for 6240 songs[4], resulting in a lyrics dataset consisting of 38,000 pairs, and an audio dataset consisting of 1000 pairs. We split both datasets into training and validation sets, and we ensured that both sets have an equal number of positive (similar) and negative (not similar) pairs.
We stored the audio samples as .wav files and then converted them into NumPy arrays using SciPy [Jones et al., 2001–]. We then split the data into blocks of 11025 values (half of the sampling rate of all the files in our dataset). We then computed the absolute value of the Fast Fourier Transform of this data, and used this as input to our model.

4.2 Experiments: Lyrics-based

The first set of experiments we ran, answer the following question: "*Can song lyrics predict song similarity?*". Each song is represented by the words in its lyrics (after excluding infrequent and overfrequent words). Given a pair of songs, the lyrics of each song are passed through the same LSTM (initial cell and hidden state for each song is zero) and the two final hidden state are recorded (h_1 and h_2). h_1 and h_2 are then combined in one of four ways ways as explained below to give a a resultant representation h_{final}. h_{final} is then passed through one or two fully-connected layers (if

[1]Last.fm dataset, the official song tags and song similarity collection for the Million Song Dataset, available at: http://labrosa.ee.columbia.edu/millionsong/lastfm

[2]https://developer.musixmatch.com/

[3]http://developer.7digital.com/

[4]The 7digital API has a lower coverage of the MSD than musixmatch does, and thus we were unable to collect nearly as many examples for the audio-based model as for the lyrics-based model.

Table 1: Each partition of the examples also contained the flipped example. i.e. if ($Song_a$, $Song_b$, $class_x$) is present in the train, validation or test; then ($Song_b$, $Song_a$, $class_x$) is also added to the same set. This doubles the size of our dataset and achieves a major sanity check for the model.

	Train	Validation	Test
# positive examples	38,000 x2	8,000 x2	8,000 x2
# negative examples	38,000 x2	8,000 x2	8,000 x2

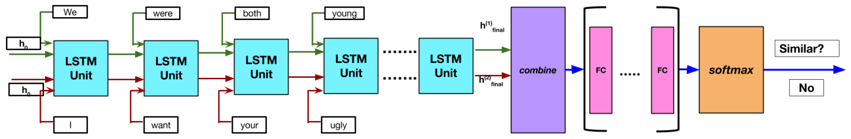
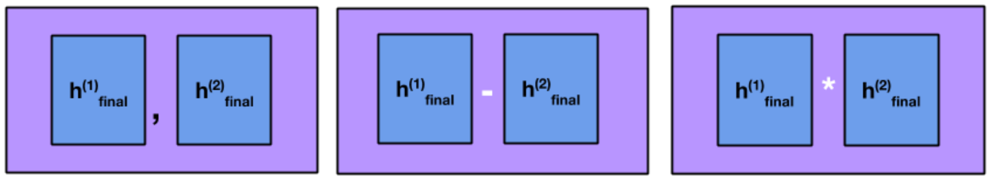


Figure 2: Our proposed model that performs similarity classifications for pairs on songs based on the lyrics of the song. Each word in the lyrics is represented as word embedding and fed as input to the LSTM at each time step.


$h^{(1)}_{final}$, $h^{(2)}_{final}$
$h^{(1)}_{final}$ - $h^{(2)}_{final}$
$h^{(1)}_{final}$ * $h^{(2)}_{final}$

(a) Concatenation $h_{final} = [h_1, h_2]$.

(b) Subtraction $h_{final} = [h_1 - h_2]$.

(c) Elementwise Product $h_{final} = [h_1 * h_2]$.

Figure 3: Different method of combining the two final LSTM hidden states obtained from running each song in a pair, each through teh same zero-initialized LSTM.

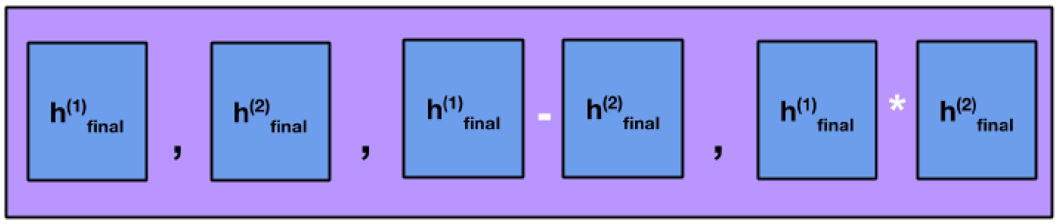

$h^{(1)}_{final}$, $h^{(2)}_{final}$, $h^{(1)}_{final}$ - $h^{(2)}_{final}$, $h^{(1)}_{final}$ * $h^{(2)}_{final}$

Figure 4: All three combinations of h_1 and h_2 from Figure 3 are used in concatenation. i.e. a concatenation of the two individual final LSTM hidden states, the difference of the two and the elementwise product of the two. $h_{final} = [h_1, h_2, h_1 - h_2, h_1 * h_2]$.

two FC layers, then 1^{st} has ReLU non-linearity) to finally give two class scores. Softmax function is used on the two class scores and cross-entropy loss is calculated.

For regularization, we use dropout ($prob\_active = 0.9$). Dropout is used between the word input and LSTM cell state, after LSTM hidden state and after the penultimate FC layer (in the case of two FC layers).

For all experiments, the word embeddings were trained from scratch. Word embeddings had a dimension of 20 and the hidden states of the LSTM had a dimension of 10 (d_h).

To get h_{final}, we are inspired by intuition and by Mou et al. [2015]. They mention three different methods of combining representation vectors of objects for comparing them. We use all three and a fourth method, which is simply a concatenation of the results of all three vectors.

Table 2: The 1^{st} column is the Combination Method used on h_1 and h_2, the final hidden states coming from the two songs in a pair. The 2^{nd} column is the number of Fully-Connected Layers used at the head of the network (taking the combination of h_1 and h_2 as input).

Combination Method	# FC Layers	Train Acc.	Val Acc.
concatenation	1	82.7%	77.8%
difference	1	49.9%	50.2%
element-wise product	1	88.6%	81.5%
all	**1**	**89.6%**	**82.8%**
concatenation	2	83.7%	78.3%
subtraction	2	85.2%	79.6%
element-wise product	2	88.4%	81.4%
all	**2**	**91.2%**	**84.7%**

4.2.1 Lyrics 1/4: Concatenation: $h_{final} = [h_1, h_2]$

h_{final} has dimension equal to $2 * d_h$. Note that in this definition h_{final} is not symmetric. ($Song_a$, $Song_b$) and ($Song_b$, $Song_a$) will produce different h_{final} vector. This case was the primary motivation for having both ($Song_a$, $Song_b$, $class_x$) and ($Song_b$, $Song_a$, $class_x$) in the dataset. Else, the network might overfit to the order in which the lyrics data is presented to it. As shown in Table 2, Concatenation proves to be the worst method of all methods. This is not too surprising since all the other methods 'add' information in some form by performing some computation on the two vectors which has a comparative nature.

4.2.2 Lyrics 2/4: Subtraction: $h_{final} = [h_1 - h_2]$

h_{final} has dimension equal to d_h. Not only is this definition not symmetric, it is anti-symmetric! Table 2 shows that with just 1 FC layer, this achieves no results! In hind-sight, this is to be expected because with just a single FC-layer, $h_{final} = x$ and $h_{final} = -x$ produce the exact opposite softmax outputs. Since h_{final} is anti-symmetric in this case and we are including both ($Song_a$, $Song_b$, $class_x$) and ($Song_b$, $Song_a$, $class_x$), we are then asking $h_{final} = x$ and $h_{final} = -x$ to generate a softmax output of 1 for $class_x$ which is obviously not possible. And the network then optimizes by giving the same weights to both classes for all cases leading to a 50% accuracy. In the case of two FC layers, $h_{final} = x$ and $h_{final} = -x$ no longer produces the same softmax output which reflects in the validation accuracy of 79.6% as shown in Table 2.

4.2.3 Lyrics 3/4: Elementwise Product: $h_{final} = [h_1 * h_2]$

h_{final} has dimension equal to d_h. This is symmetric. In Table 2, we see that a single FC layer produces a validation accuracy of 81.5% and two FC layers give 81.4%. This seems to suggest that most of the comparitve information between the two vectors is included in the elementwise produce operation and hence the addition of the second FC layer does not help the network perform much better. Indeed, of the three individual combination methods, this is by far the best.

4.2.4 Lyrics 4/4: All three: $h_{final} = [h_1, h_2, h_1 - h_2, h_1 * h_2]$

h_{final} has dimension equal to $4 * d_h$. This is not aymmentric or anti-symmetric. Table 2 shows that this has the best results. Singe FC layer gives a validation accuracy of 82.8% and two FC layers give 84.7%. When using all three methods together, the biggest concern is overfitting due to a large number of paramters. But note that our dimensionality (embedding size of 20 and hidden vector size of 10) (even 4*10=40 which is a small number in the FC layers) is very small compared to our data size (152,000 examples).

4.3 Experiments: Audio-Based

For the audio-based model, we experimented with several configurations of the convolutional layers, as well as different combinations of the final hidden states for each song. Table 3 shows the best results of our model after tuning hyperparameters.

Table 3: The 1^{st} column is the number of convolutional layers used. The 2^{nd} column is the size of the filters used, and the third column contains the hidden state combination type.

# Conv layers	# Filter size(s)	Combination	Train Acc.	Val Acc.
0	None	concatenate	63.4%	65%
0	None	subtraction	64.6%	62.3%
1	11x1	concatenate	68%	69.2%
1	**11x1**	**subtraction**	**78.2**	**76.5%**
2	11x1, 7x1	subtraction	72.4	70.1%

4.3.1 Hidden state combinations

We experimented with two combinations of the hidden states for the audio-based model: concatenation and absolute difference. These combination methods are identical to those described for the lyrics-based model in the previous sections.

4.3.2 Convolutional layers

We experimented with various settings of the convolutional layers by varying the number of convolutional layers and the size of the filters. We used the filter sizes from [Amodei et al., 2015] as a starting point.

4.4 Quantitative Results

Table 3 shows the results of our experiments with the audio-based LSTM. We see that the results of this model are very poor with the most basic approach, which uses no convolutional layers and simply concatenates the final hidden states of each song together, feeding the concatenated vector as input to the final fully connected layer. We see that using the absolute difference of the hidden states as input to the FC layer somewhat improves our results on the training set. However, using this combination method in addition to performing 1-D time-based convolutions on our input has the most impact on our model - we achieve our best training and validation accuracy of 78.2% and 76.5% with this model. The results shown in the table are from the model that uses convolutional filters of size 11x1 - this size was determined with hyperparameter tuning.
We also experimented with using multiple convolutional layers instead of one, and experimented with different sizes of the convolutional filters. However, we found that using multiple convolutional layers didn't significantly improve performance; in fact, our best model that used two convolutional layers had a lower training and validation accuracy than our best model that used a single convolutional layer. Intuitively, it seems as though using multiple convolutional layers should improve performance; we note that we didn't use max-pooling or any other techniques that are typically used with convolutional layers, and hypothesize that using these could improve performance for future iterations of this model.

4.5 Qualitative Results

Despite our initial intuition that audio might be a stronger predictor of song similarity than lyrics, our experiments here seem to indicate the opposite. One reason for this could be that our model was trained on very little data; our training set comprised of only 1000 training examples, and it is possible that our model was not able to learn enough from this training set. Further, it is possible that the audio spectrogram representation that we used is not the best possible input representation; it would be constructive to conduct experiments on the pure audio waveform (which we would expect to perform worse), as well as on more complicated representations like power-normalized spectrograms (as used in Amodei et al. [2015]).

5 Conclusion

The lyrics based approach alone shows 84.7% validation accuracy and raw audio shows 76.5%. There is room for improvement in these individual approaches as we have had limited time to work on it and for computation. Combining the two methods should definitely improve the performance.

Using difference and dot product of the song representation vectors (final LSTM hidden state) help improve the performance by a lot, both the lyrics model and the audio based model. This shows that there is information added in those operations.

For the lyrics model, second FC layer does not add much and similarly for the audio model, it is clear that using one convolution layers is better than using two convolutional layers. This shows both the power of the combination operations to capture most of the information and that music can be identified by looking at the immediate neighbourhood in the time-domain.

Our results show enough proof that using Lyrics and Audio alone is enough to classify songs as similar or dissimilar. Merging the two information sources and producing a joint model is our Future Work and we are optimistic about the results that we will obtain from it. Especially since the two information streams present unique information.

References

Algorithmic music recommendations at spotify. `http://www.slideshare.net/MrChrisJohnson/algorithmic-music-recommendations-at-spotify/22-Alternating_Least_Squarescode_httpsgithubcomMrChrisJohnsonimplicitMFMonday_January`. Accessed: 2016-04-29.

2013.

Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*, 2015.

Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

Sander Dieleman and Benjamin Schrauwen. Multiscale approaches to music audio feature learning. In *14th International Society for Music Information Retrieval Conference (ISMIR-2013)*, pages 116–121. Pontifícia Universidade Católica do Paraná, 2013.

Sander Dieleman and Benjamin Schrauwen. End-to-end learning for music audio. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6964–6968. IEEE, 2014.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`.

Qiuqiang Kong, Xiaohui Feng, and Yanxiong Li. Music genre classification using convolutional neural network.

Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. Recognizing entailment and contradiction by tree-based convolution. *CoRR*, abs/1512.08422, 2015. URL `http://arxiv.org/abs/1512.08422`.

Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Algorithmic Aspects in Information and Management*, pages 337–348. Springer, 2008.