Music Playlist Recommender System

Zhenan Fan
University of British Columbia
zhenanf@cs.ubc.ca

Wen Xiao
University of British Columbia
xiaowen3@cs.ubc.ca

ABSTRACT

With the development of information technology and Internet resources, many useful information is covered by a lot of irrelevant information. Users often spend time and energy without finding anything. In view of the phenomenon of overflowing information, personalized recommendation technology emerges as the times require. Music recommendation system provides the users with the songs that may be in line with their interests based on the content characteristics of music and the behavior of the users. Most recent papers only analyze user's preference over songs, but music is usually in context of sequence. In this work, we present a music playlist recommendation system, which not only consider the user's preference over songs but also their preference over song transitions. The main idea is to use Markov Decision Process to model the recommendation procedure, and use convolutional neural network to capture the features in song transitions. What's more, we use reinforcement-learning framework to solve this optimization problem and use Monte Carlo Tree Search to reduce the exploration time.

1. INTRODUCTION

In recent years, with the development of information technology and Internet, the music industry has shifted more to online music store and streaming services, such as Itunes, Spotify and Google Play. As a result, users have much more choices than before, then music recommender system becomes necessary, since it helps users discovering music that match their taste, and meanwhile, it helps the streaming services to target the right audience. For example, only few years ago, all the recommended music were from friends or radio, and if we would like to listen to it, we have to find the corresponding CD in store. But things get easy now, we can just search for the music on the platform or online store, we even do not need the recommendations from friends, since we have a new friend who knows us better, and his name is "music recommendation system".

In general, when most of us listen to music, we usually don't listen to a single piece of music, instead, we would like to listen to music in sequence. Therefore, it would be better if your new friend can directly recommend music in sequence, which we call playlist. Recommending playlist is not just recommend songs one by one individually, since the order always matters, to be more specific, we may like song A, song B and song C independently, but after we listen to song A, we would prefer to listen to song D instead.

Therefore, the problem we are solving is to recommend personalized playlists based on the play history of users. The task is not trivial, since there is a large semantic gap in music, and the preferred order may depend on various factor, like personality, personal experience, emotion and etc. In this paper, we introduce a novel way to predict a playlist for a given user based on his play history. Our contributions are as follows. First, we formulate the playlist recommendation problem as a Markov Decision Process with trivial transition function. Second, we formulate the user's reward function over playlist as a summation of his preference over songs and his preference over song transitions. Third, we train a convolutional neural network to capture the features of song transitions. Finally, we use a a reinforcement-learning based approach to solve this problem and use Monte Carlo Tree Search make the learning process more efficiently. We test our algorithm in the experiment section. And the result shows that our algorithm does take users preference over transitions into account.

The reminder of this paper is organized as follows. In Section 2, we introduce some related work about music playlist recommendation. In Section 3, we introduce the data set we use in this project. In Section 4, we discuss how we formulate this problem as a markov decision process and how we model the reward function. We also introduce the convolutional neural network framework we use in this problem. In Section 5, we present the reinforcement-learning based approach we use to solve the problem. In Section 6, we show our experiment results. In Section 7, we discusee the conclusion and future work.

2. RELATED WORK

Recommender system has been a popular topic in the field of Artificial Intelligence over the past decades, it is usually classified into the following categories: content-based, collaborative filtering and hybrid[2]. It is widely applied in many fields, like movies[4], traveling[9] and news[1].

For the music recommendation, the collaborative filtering methods generally outperform content-based methods, due to the semantic gap of music[12]. But Oord et al. proposed

© 2019 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

a novel content-based method for music recommendation, which solves the cold start problem. They use Convolutional Neural Network to predict the latent factor vectors for songs only based on the music audio, it provides a possible deep learning way to recommend music[14].

As Baccigalupo et al. stated, playlist is not merely a bunch of songs, but a selected collection of songs, arranged in a meaningful sequence[5]. Thus it is important to find the inner relationship between consecutive songs, Hsu et al.proposed a Convolutional Neural Network based way to predict playlists, which is mainly used in NLP[7]. They are the first to apply CNN to next-song recommendation, and showed it outperform the non-NN based ones. They inspire us that CNN is an appropriate way to invesgate the implicit relationship of songs.

Besides the novel CNN method, the traditional way to predict playlists is Markov Decision Process[11][8][10]. Hu et al. used the users' feedback information to design the reward of each song, so it is kind of collaborative filtering way[8]. To be more efficient, they make song clusters, instead of choosing a song and compute the reward of the specific song, they choose a cluster of song and compute the reward of the cluster. After deciding the next cluster, they will choose a song in that cluster based on user's preference. It will save much more time since the action space shrink a lot.

Liebman et al. [11] introduced a different representation of songs, they use the audio features to represent a song, and the reward of a song is decided by both user's preference of the song and user's preference of the transition. The transition reward is represented as the weighted sum of utilities of current song and every previous songs. As for the learning process, both Liebman and Hu uses reinforcement learning, and in Liebman's latest paper, they improved the model by applying Monte Carlo Tree Search[10].

3. DATA

We currently use the Yes playlist dataset provided by Cornell University, the data mainly collected from Yes.com and Lastfm.com. There are over 3000 songs and 1.3 million playlists in total with the most popular 250 tags. To make the data more applicable, we make some pre-processing on the data.

First, we discard the songs with number of tags less than 3, and delete them from the playlists. Generally, the playlist should not be too short or too long, so we only keep the playlists with length in the range of $[20, 30]$. After that, there remains 2367 songs and 595 playlists. Besides, the number of tags for each song varis from 3 to 81.

4. MODEL

4.1 Markov Decision Process

We want to model the music playlist recommendation problem as a Markov Decision Process (MDP). A MDP is a tuple $M = (S, A, P, R, T)$, where

1. S is the set of states
2. A is the set of actions
3. $P : S \times A \times S \to \mathbb{R}$ is the transition probability function
4. $R : S \times A \to \mathbb{R}$ is the reward function

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

Figure 1: Markov Decision Process

5. T is the terminal set, namely when the agent enters the terminal set, the process stops.

The dynamic process of MDP are as follows: an agent in the initial state s_0, and then select an action a_0 from the A. After the implementation, agent transfers to the next state s_1 according to the probability $P(s_0, a_0, s_1)$. Then an action a_1 is executed, and the agent transfer to s_2. This process ends when entering the terminal set. We can use the following diagram to show the process of state transition.

In our problem, let $\mathcal{M}$ be the whole song space. Then we can define the following MDP:

1. State space S is the set of all playlists with length smaller than l, namely
$$S = \{(a_1, \dots, a_i) : 1 \le i \le l; \forall j \le i, a_j \in \mathcal{M}\}$$
2. The set of actions A is the selection of the next song toplay, namely $A = \mathcal{M}$
3. S and A induce a deterministic transition function P. Specifically,
$$P((a_1, \dots, a_i), a) = (a_1, \dots, a_i, a)$$
4. $R^u(s, a)$ is the utility the current listener u derives from hearing song a when in state s.
5. $T = \{(a_1 \dots, a_l)\}$: the set of playlists of length l.

4.2 Song Representation

In our data set, for each song, we have its tag list. Since we have 250 tags in total, we just represent each song as a vector with length of 250, each entry is an indicator showing whether the song has the corresponding tag.

From now, for each song $a \in \mathcal{M}$, we use $\theta(a)$ to denote its binary tag representation.

Indices	Tags
1	rock
3	pop
18	classic
19	guitar
25	dance
34	country
43	sexy
46	metal
89	hip hop
92	blues
98	electronic
223	discoo

Table 1: Some sample tags

4.3 Transition Representation

To model the transition process in playlist, we want to use the convolutional neural network, since the convolutional

neural network is widely used to investigate the inner relationship between pixels(CV) or words(NLP). And it is also recently applied on the playlist recommendation task [7]. Instead of using trained CNN directly, we only use the result of convolutional layer. And we don't only consider the previous song, but the previous i songs.

4.3.1 Dimension Reduction

Before putting data into the convolutional neural network, we first do the dimension reduction. Assume we have a song sequence $a_1, \ldots, a_i$, then we can get their corresponding binary representation $\theta(a_1), \ldots, \theta(a_i)$.

First, we calculate the similarity between these binary song vectors $\theta(a_1), \ldots, \theta(a_i)$, we use Cosine Similarity to define the pairwise similarity:

$$Sim_{a_m,a_n} = \frac{\theta(a_m) \cdot \theta(a_n)}{|\theta(a_m)||\theta(a_n)|}$$

This makes sense because cosine similarity is the cosine value of the angle between two vectors. Compared with Euclidean distance, the cosine similarity pays more attention to the difference between the two vectors in the direction. And since in our case, $\theta(a)$'s are binary vectors, cosince similarity should be more relevant.

Next, we define the pairwise distance by

$$\delta_{a_m,a_n} = \frac{2 \cdot cos^{-1}(Sim_{a_m,a_n})}{\pi}$$

Finally, we use Multidimensional Scalling (MDS) to do the dimension reduction[6]. What MDS do is it takes the distance matrix as input:

$$\Delta = \begin{bmatrix} \delta_{a_1,a_1} & \cdots & \delta_{a_1,a_i} \\ \vdots & \ddots & \vdots \\ \delta_{a_i,a_1} & \cdots & \delta_{a_i,a_i} \end{bmatrix}$$

Then it can find smallest N, such that we can find $x_1, \ldots, x_i \in \mathbb{R}^N$ with $|x_n - x_m| = \delta_{a_n,a_m}, \forall n, m$, and these points $x_1, \ldots, x_i$ is the output.

When talking about dimension reduction, people natually think about PCA. Although both PCA and MDS can do the dimension reduction, they are different in: PCA minimizes dimensions, preserving covariance of data while MDS minimizes dimensions, preserving distance between data points. In our case, since we more care about the pairwise relationship between songs, it's more reasonable to use MDS instead of PCA.

We provide the 3d plot of the result from MDS in figure 2.

4.3.2 Convolutional Neural Network

The structure of CNN is shown in Figure 3. First, we use a convolutional layer to capture the inner relationship between adjancent songs. To formalize, after the demension reduction, we will have a $i \times N$ matrix X, where

$$X = \begin{bmatrix} x_1^T \\ \vdots \\ x_i^T \end{bmatrix}$$

Then we set the size of convolution kernel to be $2 \times N$, namely the kernel will analyze the inner relationship two by two. After the convolution, we will have a vector with length

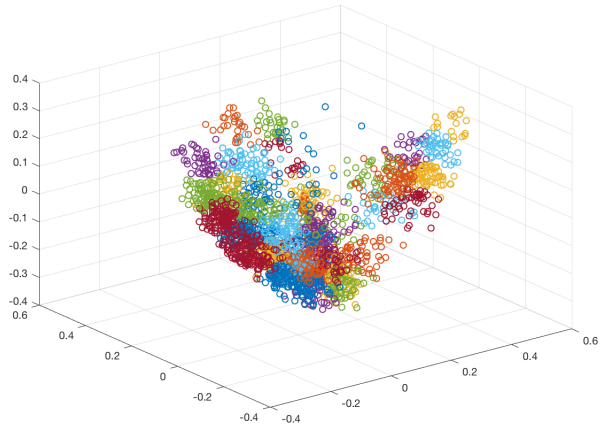

Figure 2: Multidimensional Scalling 3D plot

$i-1$. Since convolution kernel is used to capture features, we can use multiple convolution kernels to capture different features.Therefore, if we use T convolutional kernels in practice, after the convolution, we will have a vector with length $(i-1)T$.

Secondly, we use K-nearest neighborhood to divide songs into K clusters, which we will use as the label of CNN. Specifically, the output layer is equipped with the softmax activation function and the output is a distribution vector of length K, where each entry corresponds to the probability that the next song to be played belongs to that cluster.

To train the CNN, first we extracted the data from our current dataset. For every playlist, we use every $i+1$ consecutive songs, in which we use the first i songs to represent the transition, and the cluster of the last song to be the label.

After the training, we will have T trained convolutional kernels $\mathcal{K}_1, \ldots, \mathcal{K}_T$, and we use these kernels to to estabilish the transition representation:

$$\begin{aligned} (a_1, \ldots, a_i) &\Longrightarrow_{\text{Song representation}} (\theta(a_1), \ldots, \theta(a_i)) \\ &\Longrightarrow_{\text{MDS}} (x_1, \ldots, x_i) \\ &\Longrightarrow_{\mathcal{K}_1,\ldots,\mathcal{K}_T} \theta(a_1, \ldots, a_i) \end{aligned}$$

From now, for each state $(a_1, \ldots, a_i) \in S$, we use $\theta(a_1, \ldots, a_i)$ to denote its transition representation.

4.4 Reward function

The reward funcion $R^u((a_1, \ldots, a_i), a)$ is the utility user u obtains by listening song a after a song sequence $a_1, \ldots, a_i$ been listened already. We think the utility mainly comes from two parts:

- How much does user u like this song a. And we this this quantity can be expressed as a weighted sum of the song tags. Namely, if we use $R_1^u(a)$ to denote this term, then we have

  $$R_1^u(a) = \omega_1^u \cdot \theta(a)$$

  where each entry of ω_1^u denotes how much user u likes this tag.

- How much the song history $a_1, \ldots, a_i$ can affect. Recall that we use convolution kernels to calculate the inner

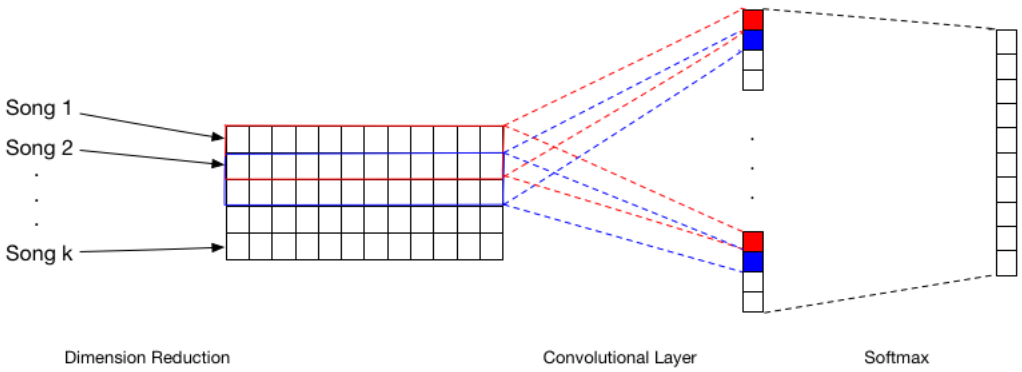

Figure 3: Structure of CNN

relationship inside the song sequences. So if we use $R_2^u(a_1, \ldots, a_i, a)$ to denote this term, we can write it as:

$$R_2^u(a_1, \ldots, a_i, a) = \omega_2^u \cdot \theta(a_1, \ldots, a_i, a)$$

where each entry of ω_2^u denotes how much user u likes this transition.

To sum up, the reward function $R^u((a_1, \ldots, a_i), a)$ can be written as:

$$\begin{aligned} R^u((a_1, \ldots, a_i), a) &= R_1^u(a) + R_2^u(a_1, \ldots, a_i, a) \\ &= \omega_1^u \cdot \theta(a) + \omega_2^u \cdot \theta(a_1, \ldots, a_i, a) \end{aligned}$$

And now our goal is to learn the user specified weights ω_1^u and ω_2^u.

5. REINFORCEMENT LEARNING

The full learning and recommending procedure is shown in algorithm 1. For user u, we first get his listening history $a_1, \ldots, a_m$, and we will recommend him with a playlist of length k, $a_{m+1,\ldots,a_{m+k}}$. Instead of selecting songs from the whole song space, we first use the similarity between playlists to find the most similar 5 or 10 playlist, and only consider the songs in the certain playlists, we define the set of songs to be candidate set M, this is kind of application of collaborative filtering. This step will save much more space and time.

First, we do the initialization of ω_1^u and ω_2^u based on $a_1, \ldots, a_m$. The detail is provided in section 5.1. Then we iteratively select song $a_{m+1,\ldots,a_{m+k}}$ in the candidate set M by Monte Carlo Tree Search, which we will discuss in section 5.2. And each time we select a new song, we will update the weight correspondingly (section 5.3).

Algorithm 1 Full Structure

```
Input: a_1, ..., a_m - the first m songs in the playlist
K - the number of songs to be predicted
M - candidate set of songs
CNN - the pre-trained CNN
Initialize ω_1 and ω_2
for j = 1 : K do
    Select song a_{m+j} from M
    Obtain reward r_j
    Update ω_1 and ω_2
```

5.1 Initialization

The procedure of initialization is shown in algorithm 2.

- First, we set all the entries of ω_1 and ω_2 to be equal weight.
- Secondly, we update ω_1 by iteratively adding $\theta(a_1), \ldots, \theta(a_m)$. This procedure actually updates user's preference over song tags based on his listening history. And we normalize ω_1 at the end.
- Finally, we update ω_2. We set a window of size i, and we iteratively adding $\theta(a_j, \ldots, a_{j+i})$ to ω_2. This procedure actually updates user's preference over transitions based on his listening history. And we normalize ω_2 at the end.

Algorithm 2 Initialization

```
Input: a_1, ..., a_m - the first m songs in the playlist
set all entries in ω_1 to be 1/len(ω_1)
set all entries in ω_2 to be 1/len(ω_2)
for k = 1:m do
    ω_1 = ω_1 + θ(a_k)
ω_1 = ω_1 / sum(ω_1)
for j = 1 : (m − i) do
    ω_2 = ω_2 + θ(a_j, ..., a_{j+i})
ω_1 = ω_1 / sum(ω_1)
```

5.2 Song Selection

Now we want to select next song to play. Although we can simply select the song a giving the highest reward $R(s, a)$, the immediate return function, $R(s, a)$, cannot tell whether the strategy is good or bad in a long term. Therefore, we need to consider the long-term utility:

$$V_\pi(s) = \sum_{i=1}^{h} \gamma^i r_i$$

where $r_i = R(s_i, \pi(s_i)), s_{i+1} = P(s_i, \pi(s_i)), s_1 = s$

And we select next song a by

$$\pi^* = \arg\max_\pi V_\pi(s), a = \pi^*(s)$$

Since song space is very big, it's impossible to exhausts all the possibilities. Therefore, we need to use some other method to find π^*. Traditionally, people will use Monte Carlo method to sample many song sequences, and then pick one giving highest reward. However, if random moves are selected for each round, it's hard to find the best way to move forward. Therefore, we decide to use Monte Carlo Tree Search to do the song selection[3].

Monte Carlo Tree Search is a method of making optimal decision in the problem of artificial intelligence, usually in the form of action (move) planning in a combined game. It combines the generality of random simulation and the accuracy of tree search.

The sturcture of the Monte Carlo Tree Search is shown in the figure 4:

Monte Carlo Tree Search consists of four main steps: Expansion, Simulation, Back Propogation and Exploitation.

1. **Expansion**

   If current node is not a terminal node, namely the playlist length is not maximal, then we randomly select a new song and set to be the child of the current node.

2. **Simulation**

   Start simulating a song sequence from the current node until it reaches maximal length.

3. **Back Propogation**

   Every node n_i is associated with a touple (T_i, v_i), where T_i is the numer of visit through n_i, namely the number of times n_i is selected, and v_i is the average value of all the simulation results of a subtree with node n_i as the root node. More specifically,

$$T_i = \sum_{n \in Children(n_i)} T_n$$

$$v_i = \frac{\sum_{n \in Children(n_i)} v_n T_n}{T_i}$$

   where $Children(n_i)$ is the set of all the children of n_i. That is, the number of access times of the parent node, T_i, is the sum of the number of access for all the children, and the observed return value v_i is the weighted avarage of all the returns of the child nodes.

   Then when the leaf node obtains the new v value and T value through simulation, we updates the v values and T values of all the internal nodes on the search path.

4. **Exploitation**

   Start from current node n, for every child n_i we give it a evaluation value r_i, and we select the child with highest r_i, where

$$r_i = v_i + c\sqrt{\frac{\ln(T_n)}{T_i}}$$

   where c is the exploration parameter used to balance exploration and exploitation. As we can see, v_i corresponds to exploitation, namely the node with high possible return are more likely to be selected. And $\sqrt{\frac{\ln(T_n)}{T_i}}$ corresponds to exploration, it is high for moves with few simulations.

Algorithm 3 MCTS

```
while running time limitation not reached do
    node = root
    while node.Max_depth not reached do
        if node have children then
            if with probability p then
                node = expand(node)
                Break
            else
                node = bestchild(node.children)
        else
            node = expand(node)
            Break
    if node.Max_depth not reached then
        Simulation(node)
    Back_Propogation(node)
return bestchild(root)
```

5.3 Weights Update

After selecting song a_{m+j}, we will update the weights ω_1 and ω_2 correspondingly. The full procedure is shown in algorithm 3. r_j is the reward generated by this song.

- First, we let $\bar{r}$ be the average of history rewards $r_1, \ldots, r_{j-1}$
- Secondly, let $r_{direction} = \log(\frac{r}{\bar{r}})$. Note that $r_{direction} > 0$ if $r_j > \bar{r}$ adnd $r_{direction} < 0$ if $r_j < \bar{r}$. Therefore, $r_{direction}$ determines the direction of the update. What's more, the farther the distance between r_j and $\bar{r}$, the greater the value $r_{direction}$. Thus, $r_{direction}$ also determines the strength of the update.
- Finally, we update ω_1 and ω_2 by

$$\omega_1 = \frac{j}{j+1} \cdot \omega_1 + \frac{1}{j+1} \cdot \frac{R_1}{R_1 + R_2} \cdot r_{direction} \cdot \theta(a_{m+j})$$

$$\omega_2 = \frac{j}{j+1} \cdot \omega_2 + \frac{1}{j+1} \cdot \frac{R_2}{R_1 + R_2} \cdot r_{direction} \cdot \theta(a_{m+j-i}, ..., a_{m+j})$$

Algorithm 4 WeightUpdate

```
Input: θ(a_{m+j}), θ(a_{m+j-i}, ..., a_{m+j})
{r_1, ..., r_j} rewardList - reward list
ω_1, ω_2 - parameter in reward function
Let r̄ = mean({r_1, ..., r_{j-1}})
r_direction = log(r / r̄)
ω_1 = j/(j+1) · ω_1 + 1/(j+1) · R_1/(R_1+R_2) · r_direction · θ(a_{m+j})
ω_2 = j/(j+1) · ω_2 + 1/(j+1) · R_2/(R_1+R_2) · r_direction · θ(a_{m+j-i}, ..., a_{m+j})
```

6. EXPERIMENT

Instead of selecting songs from the whole song space, we first use the similarity between playlists to find the most similar 5 or 10 playlist, and only consider the songs in the certain playlists, we define the set of songs to be candidate set, this is kind of application of collaborative filtering. This step will save much more space and time.

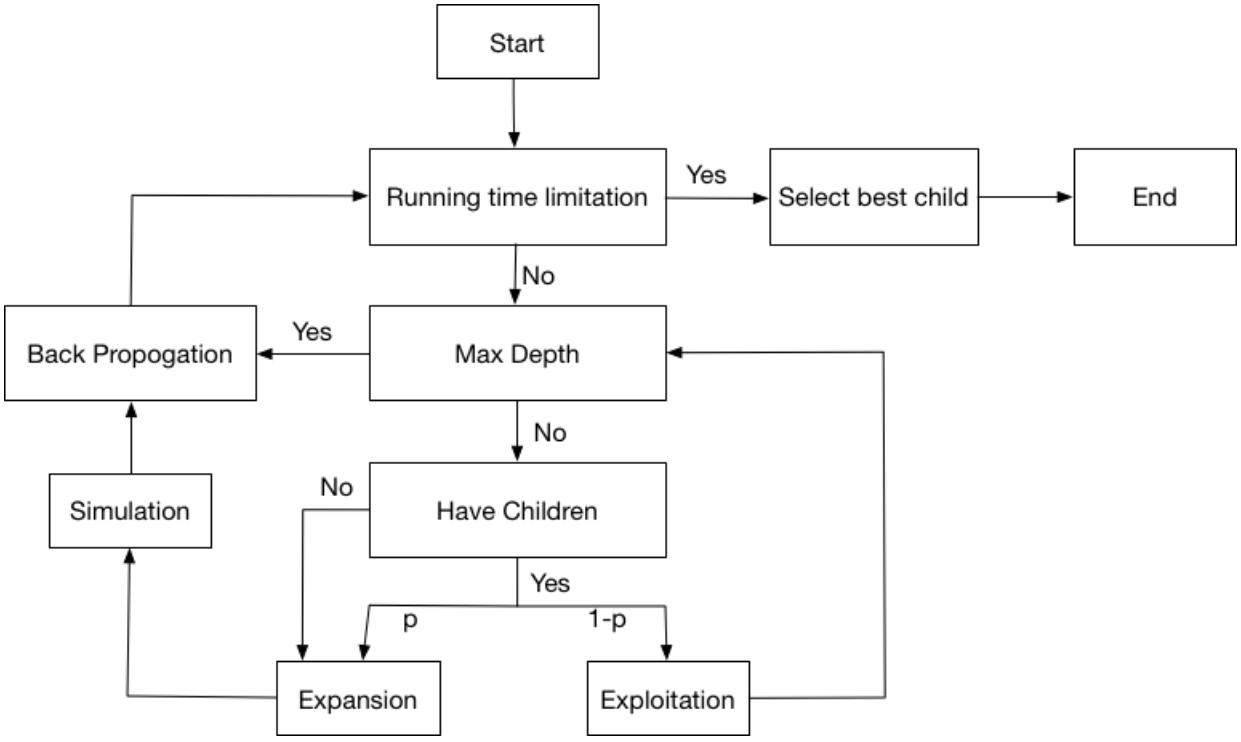

Figure 4: Monte Carlo Tree Search

6.1 CNN

We set the transition length i to be 5, which means for the song a_k, we will consider the previous 4 songs, i.e $a_{k-4}, ..., a_k$ as the input of CNN, and the cluster of a_{k+1} is the label. Then we have 11,116 data in total, and then we split the data into training set, validation set and test set. As for the kernel, we set 10 kernels in total, thus the result of convolutional layer is a vector of length 40.

Besides, we use the Theano[13] to train the CNN on GPU, which will make it more efficient.

After training, the accuracy of test set and train set is around 17% with 100 classes.

6.2 Baseline

We set two baseline algorithm, the first one is just randomly pick songs from the candidate set. The second baseline is the greedy algorithm, by setting this baseline, we want to show that the order really matters, the sequence we generated is better than picking the song with largest reward individually each time. After selecting the song with largest reward, the parameter in reward function will be updated the same as in our CNN-MCTS method.

6.3 Comparison

For the evaluation, we use two methods: one-to-one similarity and max similarity.

The one-to-one similarity is to measure the similarity of prediction and true playlist one by one, i.e

$$Sim1 = \sum_{i}^{num\_pred} sim(pred[i], playlist[i])$$

The max similarity is to measure the max similarity of prediction and all the remaining songs in the playlist, i.e

$$Sim2 = \sum_{i}^{num\_pred} \max_{j} sim(pred[i], playlist[j])$$

The one-to-one similarity mainly measures the transition preference of the user and the max similarity mainly measures the song preference of the user.

Table 2: Result comparison with number of prediction equal to 3

	Mean one-to-one similarity	Mean max similarity
Random	0.63	1.13
Greedy	0.91	1.75
CNN-MCTS	1.28	1.83

The experiment results are shown in Table 2 and Figure 5. On one hand, based on the result, we can see that when using max similarity, the CNN-MCTS algorithm has similar results as Greedy. However, when using one-to-one similarity, CNN-MCTS algorithm has obviously better result than the greedy algorithm. This shows that the greedy algorithm can capture the user's preference on songs but not the preference on transitions. On the other hand, from the graph we can see that greedy algorithm has smaller variance than CNN-MCTS algorithm, this shows that our algorithm is not very stable, and this might be one of our future work.

For a more intuitive explanation, here we give an example:

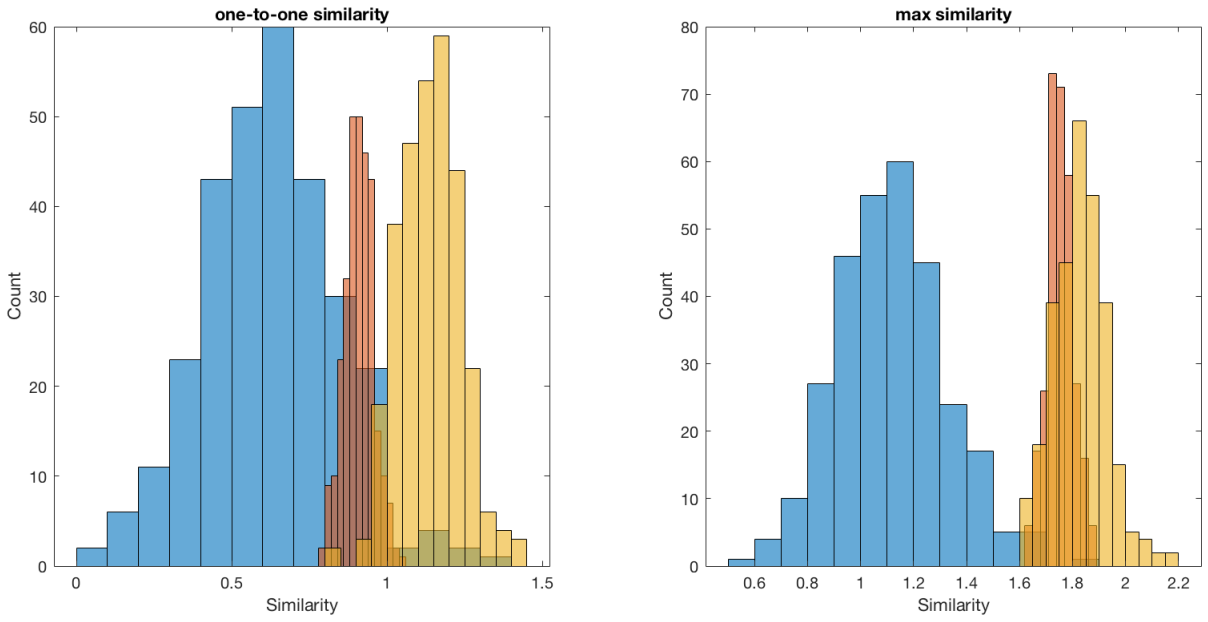

Figure 5: Comparison Results. Blue: Random; Red: Greedy; Yellow, CNN-MCTS

Table 3: An example: the first 13 songs in a playlist, and given the first 10 songs, we are predicting the next 3 songs

Order	Song Name	Artist
1	Firework	Katy Perry
2	Moment 4 Life (w/ Drake)	Nicki Minaj
3	Like A G6	Far East Movement
4	Right Above It (w/ Drake)	Lil Wayne
5	We R Who We R	Ke$ha
6	Aston Martin Music	Rick Ross
7	Knock You Down	Keri Hilson
8	Whatever You Like	T.I.
9	Only Girl (In The World)	Rihanna
10	Grenade	Bruno Mars
11	I'll Be Missing You	Puff Daddy
12	What's My Name (w/ Drake)	Rihanna
13	Hold Yuh	Gyptian

Table 4: Result of Greedy algorithm

Order	Song Name	Artist
11	Just A Dream	Nelly
12	DJ Got Us Fallin' In Love	Usher
13	Down (w/ Lil Wayne)	Jay Sean

Table 5: Results of CNN-MCTS algorithm

Order	Song Name	Artist
11	Bonnie & Clyde (w/ Beyonce)	Jay-Z
12	What's My Name (w/ Drake)	Rihanna
13	Love The Way You Lie (w/ Rihanna)	Eminem

Now we take a look at the tag list for these songs

True Playlist

1. I'll Be Missing You: ' 90s', ' rnb', ' hip hop'
2. What's My Name (w/ Drake): ' 00s', ' hip-hop', ' love'
3. Hold Yuh: ' male vocalists', ' love songs', ' party'

Playlist generated by greedy algorithm

1. Just A Dream: ' 00s', ' hip hop', ' catchy'
2. DJ Got Us Fallin' In Love: ' hip-hop', ' love', ' hot'
3. Down (w/ Lil Wayne): 'hot', ' dance', 'pop'

Playlist generated by CNN-MCTS algorithm

1. Bonnie & Clyde (w/ Beyonce): ' 00s', ' rnb', ' hip hop'
2. What's My Name (w/ Drake): ' 00s', ' hip-hop', ' love'
3. Love The Way You Lie (w/ Rihanna): ' male vocalists', 'love songs', 'rap'

As we can see, compared with the playlist given by the greedy algorithm, the one given by the CNN-MCTS algorithm has more similar song transition as the true playlist.

7. CONCLUSION AND FUTURE WORK

In this paper, we present a novel music playlist recommendation system, which not only consider user's preference over songs but also their preference over song transitions. In the experiment, we show that our algorithm can generate better playlists compared with a more traditional method which only put user's preference over songs into consideration.

For future work, we can improve the model in the following parts:

- The result depend much on the result of CNN, but the CNN model is not very stable, we would like to improve the CNN model to make it more stable. It is a good way to increase the size of dataset.
- In this paper, we use the cosine similarity to measure the similarity between songs, and we can Jaccard similarity to see if it can give better results.
- For the candidate set of MCTS, now we used the songs in the most similar n playlists, which is a way to apply the collaborative filtering. And next step we can use weighted matrix factorization instead.

8. REFERENCES

[1] Sofiane Abbar, Sihem Amer-Yahia, Piotr Indyk, and Sepideh Mahabadi. Real-time recommendation of diverse related articles. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 1–12, New York, NY, USA, 2013. ACM.

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, June 2005.

[3] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, May 2002.

[4] Amos Azaria, Avinatan Hassidim, Sarit Kraus, Adi Eshkol, Ofer Weintraub, and Irit Netanely. Movie recommender system for profit maximization. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 121–128, New York, NY, USA, 2013. ACM.

[5] Claudio Baccigalupo and Enric Plaza. *Case-Based Sequential Ordering of Songs for Playlist Recommendation*, pages 286–300. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[6] I. Borg and P.J.F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.

[7] Kai-Chun Hsu, Szu-Yu Chou, Yi-Hsuan Yang, and Tai-Shih Chi. Neural network based next-song recommendation. *CoRR*, abs/1606.07722, 2016.

[8] Binbin Hu, Chuan Shi, and Jian Liu. *Playlist Recommendation Based on Reinforcement Learning*, pages 172–182. Springer International Publishing, Cham, 2017.

[9] Asher Levi, Osnat Mokryn, Christophe Diot, and Nina Taft. Finding a needle in a haystack of reviews: Cold start context-based hotel recommender system. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 115–122, New York, NY, USA, 2012. ACM.

[10] Elad Liebman, Piyush Khandelwal, Maytal Saar-Tsechansky, and Peter Stone. Designing better playlists with monte carlo tree search. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 4715–4720, 2017.

[11] Elad Liebman, Maytal Saar-Tsechansky, and Peter Stone. Dj-mc: A reinforcement-learning agent for music playlist recommendation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, pages 591–599, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems.

[12] M. Slaney. Web-scale multimedia analysis: Does content matter? *IEEE MultiMedia*, 18(2):12–15, Feb 2011.

[13] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[14] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2643–2651. Curran Associates, Inc., 2013.

In its simplest form, the Ising Model consists of a NxN lattice of binary variables $x_i \in \{-1, +1\}$ that are locally connected horizontally and vertically with pairwise potentials. There can also be an external field applied to the variables that biases them toward a particular state. The total energy of a simple Ising model we consider here is defined as

$$E = -J \sum_{(i,j)\in E} x_i x_j - J_b \sum_{i \in V} b_i x_i$$

Where the first sum is over all edges of the lattice and the second over all nodes. J, J_b, b_i are the strength of pairwise interactions, strength of external field, and per-pixel binary desired values. The corresponding un-normalized probability distribution over states of the lattice is:

$$\pi(x) = exp\{J \sum_{(i,j)\in E} x_i x_j + J_b \sum_{i \in V} b_i x_i\}$$