The Application of Sequential Models to Session-Based Recommendation System

The exploration of the language model applied to music ranking tasks

Qiqi Zeng[1]

MSc Web Science and Big Data Analytics

Supervised by Dr. Emine Yilmaz and Dr. Rishabh Mehrotra

Submission date: 06 Sep 2019

[1]**Disclaimer:** This report is submitted as part requirement for the MY DEGREE at UCL. It is substantially the result of my own work except where explicitly indicated in the text. *Either:* The report may be freely copied and distributed provided the source is explicitly acknowledged
Or:
The report will be distributed to the internal and external examiners, but thereafter may not be copied or distributed except with permission from the author.

Abstract

The main goal of this project is to explore the application of the language model to the session-based music sequential tasks. Specifically, in this project, a recommendation system was designed to help user automatically rank the unplayed tracks in a playlist, based on the information of the played tracks. Different from the recommenders relied on user models, the models implemented in this project require no user data. This project examined the effect of three neural networks: RNN, LSTM and Trellis network [7]. The experiments result found that the Trellis network performs best overall. To introduce users' response to each tracks in a session, this project compared two different methods, which will be explained in detail in this report. In addition, two different methods of exporting model outputs will also be tested in this project. Finally, this project concludes an insightful rule on the processing of music sequence: while the non-skipped tracks can be regarded as the tracks that are liked by users, the skipped tracks are not necessary to be disliked by user.

This project was cooperated with Spotify London, and the data used in this project was from Spotifys public dataset.

Acknowledgements

Firstly, I would like to express my gratitude to my academic supervisor, Dr. Emine Yilmaz, for her supervision. Thanks a lot to my enterprise mentor, Rishabh Mehrotra, for his kind help, creative ideas and insightful suggestions during my project. I would like to thank the Spotify London for their resources and data provided. Finally, I would like to thank my father and mother for all of their supports on my study.

Contents

List of Figures **3**

List of Tables **4**

1 Introduction **5**

1.1 Motivation 5

1.2 Project Aims and Task framing 6

1.3 The Setting of Research Question 6

1.4 Application 8

1.5 Public Repositories 8

2 Background **9**

2.1 Introduce to Recommendation System 9

2.2 Sequential Model 10

2.2.1 Language Model 10

2.2.2 Traditional Recurrent Neural Network 11

2.2.3 Long Short-Term Memory 12

2.2.4 Trellis Network 14

2.3 Optimization 16

2.3.1 Dropout 16

2.3.2 Deep Supervision 17

2.3.3 Weight Normalization 17

2.3.4 Mixture of Softmax 18

2.4 Evaluation Method 18

2.4.1 Normalized Discounted Cumulative Gain 18

3 Data Analysis **19**

3.1 Data Description 19

3.2 Exploratory Data Analysis 19

4 Methodology and Experiment **23**

4.1 Feature Engineering 24

4.2 Approaches to Applying Skip Information to Model Input 24

4.2.1 Approach 1: Deleting Skipped Tracks 24

4.2.2 Approach 2: Adding Session Information 25
4.3 Neural Network Architecture 26
4.3.1 Published Deep Learning Algorithm 26
4.3.2 Optimization 26
4.4 Output and Loss Function 27
4.5 Evaluation 28
4.5.1 Ranking Method 28
4.5.2 Ranking Evaluation Metric 29
4.6 Experiment Set Up 29

5 Results and Discussion **31**
5.1 Testing on the whole test data 31
5.2 Testing on different groups of test data 33
5.3 Revelation on new ranking method 34

6 Conclusion and Future Works **36**
6.1 project conclusion 36
6.2 Further Works 36

bibliography **38**

A Supplementary Chart and Forms **42**
A.1 Session Length Distribution 42
A.2 Session Length Distribution 43
A.3 Heatmap for correlations between track features 44
A.4 Standard deviation distribution of release years within a single session 45

List of Figures

2.1 a Recurrent Neural Network [2] 11
2.2 a LSTM memory block [4] 13
2.3 The inter-layer of TrellisNet [7] 14
2.4 An atomic view of LSTM Non-linearity activation function [7] 15
2.5 Variational Dropout [23] 17

3.1 Std of Popularity for all tracks 21
3.2 Std of Popularity for minor tracks 21

4.1 The Structure of the training process 23
4.2 Deleting Skipped Tracks Approach 25
4.3 Adding Session Information Approach 25
4.4 The output of Neural networks 27

A.1 Std of Release Year for all tracks 45
A.2 Std of Release Year for minor tracks (only include sessions with at leat one track that released before 2010)) 45

List of Tables

5.1 The NDCG Score on the Whole Test Dataset 31
5.2 The NDCG score on the sessions where the number of skipped tracks is LESS than x 33
5.3 The NDCG score on the sessions where the number of skipped tracks is MORE than x 33
5.4 The NDCG scores of Improved ranking algorithm 35

Chapter 1

Introduction

Spotify is an online music streaming services provider. Since 2006, it has attracted millions of users to play music on their platform, which helps Spotify to collect abundant user behaviour data for users' music preference research. In addition to the music search, Spotify provides their music mainly by recommendation. Various algorithms have been applied to increase the recommender accuracy and thus improve user experiences. This project aims to design an algorithm that can automatically rank the future tracks that have not yet been played in a playlist, according to the response of the user to the played tracks. The ranking result should give priority to the tracks that are most likely to be appealing to users, and place the less appealing tracks to the end of the playlist. This function can greatly improve the user's streaming experience. This chapter will give a brief introduction to this project.

1.1 Motivation

Many recommendation systems today tracks users' historical behaviour data to predict user preference, and then recommend relevant content to them. In music service websites and applications, more specifically, in Spotify, the system can track users' behaviour data and record these data to enterprise database once user logs into account. When users need to get some music suggestions, the website or application can take the advantage of users historical data that they stored, apply algorithms to analyse these data, and then retrieve the most relevant songs from their music library for users.

With users' personal information, it is relatively easy to predict the music habit and preference for a single user, making it feasible to recommend contents in an accurate way. However, users' historical data is not always accessible. The most common case is that some users may not be accustomed to log into the account, or there is a new user that uses this music service for the first time. In these cases, the only valid information for the recommendation system is session information, which records user's session logs starting from the user opening the music service for this time only. Compared with the accumulated personal data for each single user, this session logs provide extremely limited information, enormously increasing the difficulty of predicting user preference. Consequently, it is necessary to apply more competent models to capture valid features more precisely from this limited information. Therefore, this project aims to design a model that can predict user preference and recommend relevant contents to users based on only session information.

1.2 Project Aims and Task framing

As mentioned above, this project aims to design a recommendation system that automatically rank the tracks which have not been played in a play list, based on the previous session logs. A positive ranking result should sort the tracks that are most likely to be liked by users before the others.

To specify the tasks in this project in detail, suppose there is a session sequence

$$S^i_{i:T} = t^i_1, t^i_2, ..., t^i_{T/2}, t^i_{T/2+1}, ..., t^i_T \tag{1.1}$$

where t_t represent a track in session i at position t. In addition to the position of each track in the session, a user's session log R is available. The session log for each track r^i_t includes the information of user's response to each track, such as if user paused before play, if the user changed context between the previous track and the current track, as well as the skip information, $k^i_t \in 0, 1, 2, 3$, where 0 means not skip any part of that track, 1 means skip a small part, 2 means skip half and 3 means skip the whole track. Whether the user likes a track can be observed from whether he skip it or not. The goal is to rank the last half of the tracks in a session $(t^i_{T/2+1}, ..., t^i_T)$ according to the sequence of the first half of that session $(t^i_1, ..., t^i_{T/2})$ and user's response to them $(r^i_1, ..., r^i_{T/2})$. The evaluation of this ranking algorithm depends on the skip information of the last half of tracks $(k^i_{T/2+1}, ..., k^i_T)$. Ideally, the last half of tracks should be ranked in order so that the skip information in that order (from the first track in the re-ordered session to the last track) increases from 0 to 3 gradually.

1.3 The Setting of Research Question

Different from the user profile-based system, the ranking tasks in this project are all session-based, i.e., there is no user information, all the predicting should only base on the users' session information. Therefore, the traditional recommender algorithm, such as collaborative filtering, could be hard to apply to this task. In addition, the limitation in valid data largely increases the prediction difficulty, because the music listening habits of different users can vary a lot, so it would be harder to capture the regular pattern of skip probability.

From this limited information, trying to get more information in addition to the musical features of each track, the order of tracks in a session could also be used as an important clue. Thus, the recommendation tasks are transformed into the sequential prediction problem. There are many neural network architecture available to solve the sequential problem, most of them are RNN-based, such as the convectional Recurrent Neural Network [1][2], Long Short-Term Memory (LSTM) [3][4] and Gated Recurrent Unit (GRU) [5]. Recently, CNN-based approach [6] is also proved to be competent for sequence tasks. Some recently proposed model even combined RNN-based approach and CNN-based approach together, such as the trellis network [7]. This project will figure out which neural network structure is the most efficient structure for the session-based recommendation.

To rank the rest of the tracks, it can be regarded as the task to determine which track in the last half of the session is most likely to be the next non-skipped track. This task is very similar to the language model [8][9] in the Natural Language Processing (NLP) problems, which aims to the find the most likely next word from the whole corpus. However, to our knowledge, language model is seldom applied to the recommendation tasks. Therefore, this project tries to figure out how language model can be used to improve the recommendation accuracy.

However, training a music sequences is not exactly same as training a sentence in the Language Model problem. When a sentence is created, the order of each word is sorted actively by speaker or writer, i.e., since the speaker or writer generously create a sentence following the language convention, the order in which words appear in this training example sequence may reappear in other sentences as well. However, the tracks sequence in a session sometimes can be ranked passively. In most session sequences, the order of tracks is pre-arranged by the music service provider, so some tracks would be skipped by a user, which means that user does not like these tracks. Therefore, unlike directly setting the next words as the training target in the language model, the style of the next track may not be what user want, and if user could positively create a track sequences in which the tracks would not be skipped by themselves, these skipped tracks in the training set would not be qualified candidates. Thus, it is unwise to straightly set the next track in the session sequence as a training target. When setting the training targets (training label), the skip information should be considered to decide if the next track in the training session is valid to predict user preference. This project will design different ways to solve this problem and compare these model results to offer the most effective way.

Many language models treat the next word prediction problem as a classification problem. By applying a decoder to the last layer of the neural network, the most likely words can be selected out from the model output using the Softmax function [10]. However, as the corpus is always extremely large (this is same for the music library), the decoder can be hard to learn with limited training data. Since the task in this project is to rank for specified tracks, this project will figure out that if the model can capture more output features by setting the model output to the same dimension of music embedding (similar to word embedding), and doing backpropagation based on the mean squared error between the model output and the music embedding of target track. This will be explained in detail in Methodology Explanation in this report.

Sorting out the problems mentioned above and the solutions to them, the final research questions in detail are listed below:

1. Which neural network architecture for session-based recommendation in the field of music? More specifically for this project, which structure perform better among conventional RNN, LSTM and trellis network?

2. Should we applied language model for Recommendation systems, and is it efficient on music ranking tasks?

3. What is the best technique to manage the skipped tracks in training sessions? Should the model be trained on all the tracks or only on the non-skipped tracks in a session?

4. Which is the best way to set the model output? What is the best ranking criterion?

1.4 Application

Although the task in this project is to rank the data for the last half of a session, however, from a macroscopic view, the models implemented in this project can also work as a recommendation system when extending the last half of a session to the whole music library, i.e., the music service provider can use these models to rank the whole music library and pick out recommended songs from the top ranked position.

The algorithms applied in this project are all session-based. They do not require any personal data of users. Ranking and recommendation only rely on the record of user operations a few second ago. Therefore, it is competent to rank or recommend music for a new user or a non-login user. Even for a user who has logged in, if he tries to find out some tracks with the music style that he didn't often play before, he can benefit from these session-based algorithms. This is important because many traditional recommendation algorithms can finally tend to suggest only one style of music to users, narrowing users' available music scope. In addition to the field of music, the output of this project can also be applied to the field such as web advertisement recommender and online commodity recommender, as they often do not require users to login, or there are laws to protect users' privacy information, causing the inaccessibly to user information.

1.5 Public Repositories

The code of this project is available on the GitHub: https://github.com/ucabqze/trellis-network-applied-to-music-sequences. The code for different models are divided into different folders, and the detailed instructions can be found in *README.md*.

Chapter 2

Background

This chapter will introduce the past related works on recommendation systems and sequential models. It will analysis their successful experience as well as their drawbacks, discuss how to improve their works and list out the relevant concepts used in this project.

2.1 Introduce to Recommendation System

The two main means for users to obtain information are search engine and recommendation system. While searching is active and goal-oriented, the recommendation is passive, and it is designed for users without explicit purposes. A well-designed recommendation system is able to provide users with the contents which are attractive, and normally match their personal preference. Thus, studying historical user behavior data and finding out an effective algorithm are crucial.

Currently, the most commonly-used algorithms for recommendation system are collaborative filtering and content-based recommender. In order to adapt to various specific domains, these fundamental algorithms were further developed. In music recommendation, for example,Zhu et al. [11] had proposed a neighbor-based collaborative filtering to deal with large dataset efficiently and effectively, and to improve the recommendation quality, they applied Discriminative Reweighting and Discriminative Reranking. However, Vandenoord et al. [12] pointed out that collaborative filtering surfers form cold star problem, so they proposed a deep content-based model, and predicted latent factors from music audio. Volkovs et al. [13] made a further adjustment to the basic models, presenting an approach of two-stage model, in which the first stage was designed for efficient retrieval and the second stage was added to re-rank retrieved candidates to improve the accuracy of top elements in the recommendation list.

However, the traditional algorithms mentioned above have some limitations. As these algorithms focus on individual difference, the largely depend on users profile. However, there are cases that websites or applications do not track users' histories, and they only recommend items according to session-based data. For those recommendation systems without long user histories, these algorithms, such as collaborative filtering, would be not effective. In addition, while they study users' personal preferences, they ignore the sequence of user history. As each item is independent of each other, it is hard for these algorithms to model the continuous preference information of items in a session.

Considering the sequencing problems, Hidasi et al.[14] replaced the traditional item-to-item recommendation, and applied a sequential model to improve the performance of session-based recommendations. They proposed an RNN-based approach to modeling the whole available session. Based on Standard RNN, they introduced Gated Recurrent Unit(GRU) [15] as the core of their network. In the last layer, they output the probability of the next item in the session for all valid items. Reference to the structure of this paper, sequential model was applied to our project. The next section will explain the concepts of sequential model in detail.

2.2 Sequential Model

The sequential model is invented from the step-by-step reaction mechanism, aims to solve problems of successively response categories [16]. From a mathematical point of view, sequential model can be defined as: given an input $x_{1:T} = x_1, \ldots, x_T$ with sequence length T, a sequential model is any function $G : \mathcal{X}^T \to \mathcal{Y}^T$ such that

$$y_{1:T} = y_1, \ldots, y_T = G\left(x_1, \ldots, x_T\right) \tag{2.1}$$

where y_t should rely on $x_{1:t}$ and not on $x_{t+1:T}$.

Thus, sequential model is widely applied to many Natural Language Processing problems, such as language modeling, machine translation and sentimental analysis. As the music tracks in a session are also step-wise, it can be argued that by applying the sequential model, a more accurate recommendation result would be provided. Specifically, for session-based tasks, as there is no user information, the music preference in each session can vary from each other. Thus, in this case, it is more crucial to apply sequential model to collect more valid information. Currently, the most commonly used neural networks that applied to sequential models comprise Recurrent Neural Network, Gated Recurrent Unit, Long Short-Term Memory and so on. This section will introduce the relevant concepts that were used in this project.

2.2.1 Language Model

One of the vital application of sequential model is language model. A statistic language model is a probability distribution that estimates the probability of the next word appeared after a certain word in a given sequence. Mathematically,

$$P\left(w_0, \ldots, w_N\right) = P\left(w_0\right) \prod_{i=1}^{N} P\left(w_i | w_0, \ldots, w_{i-1}\right) \tag{2.2}$$

where w_i represent discrete words in a corpus. In the field of music, music tracks in an online music streaming regularly come in a specific order. Take the user behaviour data into account, this order tends to follow a user habit convention. Thus, with adequate historical data, the next track in a specific music session seems to be predictable. Similar to language modeling, the algorithm in session-based recommendation tries to find the regular pattern in a historical session data to predict the next track, which can analogy with predicting the next words.

Previously, the widely applied method to language model tasks was n-gram models, which is based on the statistical distribution and conditional probability. However, this classical method can be hard to capture remote characters, and it suffers from data sparsity [8]. The work of Bengio et al. [9] introduced Neural Network to the training of language model. They stated that the traditional word representation led to the problem of dimensionality cures, and they supposed that this problem can be solved by learning the distributed representation of words. In addition to word sequence probability distribution, their model learns the word representation distribution simultaneously. Based on their work, the word embedding now can represent the features in itself, thus the similarity between each word can be estimated by calculating the distance between word embeddings, e.g., Euclidean distance, cosine distance, etc. The introduction of Neural Network allows the model to capture more information than statistical probability approach, which remarkably improves the performance of traditional n-gram model. The state of the art presently for language model is based on the structure of RNN and LSTM, which will be explained in detail in the next two subsections.

2.2.2 Traditional Recurrent Neural Network

With the ability to capture unbounded context, Recurrent Neural Network (RNN) becomes one of the most commonly used sequential neural network, and has a great contribution to language modeling. It is basic and easy to implement and train. A simple Recurrent Neural Network, which is also called Elman network [1], has been applied in the works of Mikolov et al. [2]. Their structure is showed in Figure 2.1.

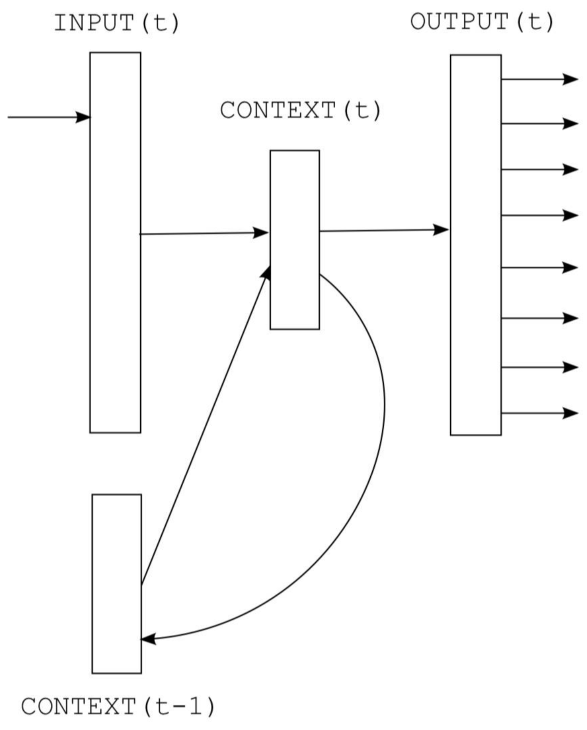


Figure 2.1: a Recurrent Neural Network [2]

As this figure shown, the network is structured into three layers: input layer, hidden layer (context layer) and output layer. In the time t, the input, hidden state and output are represented by x(t), s(t), and y(t) respectively, and the x(t) is constituted by concatenating the current word vector represented by w and the output from neurons in context layer from the last timestamp, which is represented by s(t-1). The specific formulas for each element in this network are shown below:

$$x(t) = w(t) + s(t-1) \tag{2.3}$$

$$s_j(t) = f\left(\sum_i x_i(t)u_{ji}\right) \tag{2.4}$$

$$y_k(t) = g\left(\sum_j s_j(t)v_{kj}\right) \tag{2.5}$$

where f(z) is sigmoid activation function:

$$f(z) = \frac{1}{1+e^{-z}} \tag{2.6}$$

and g(z) is softmax function:

$$g\left(z_m\right) = \frac{e^{z_m}}{\sum_k e^{z_k}} \tag{2.7}$$

Mikolov et al. [2] showed that for the language model problem, this RNN performed significantly better than the other models in their experiment, which include Lattice model, Discriminative LM and Joint LM. However, Hochreiter and Schmidhuber [3] pointed out that the Elman network suffers from the problem that error signals flowing backwards in time can lead to the problems of gradient explosion and gradient vanishing. Thus they launched the gradient-based method, long short-term memory (LSTM), which will be explained in detail in the next section.

2.2.3 Long Short-Term Memory

Using the conventional RNN, it can take a long time to learn history information over an extended time interval, due to the inefficiency of the network structure. To model temporal sequences and their long-range dependencies more accurately, Long Short-Term Memory [3] was invented. Long Short-Term Memory (LSTM) is a specific RNN architecture. To solve the problem of vanishing error signals, it introduced additional features: an input gate, an output gate and a memory cell. According to Hochreiter and Schmidhuber [3], these gates can help to avoid input weight conflicts, and the memory cell was designed to code a distributed input efficiently. The whole memory cell blocks help facilitate information storage [3], and a single memory cell block is shown in the figure 2.2 for clarity.

The mathematical formulas below describe the figure 2.2 clearly. In these formulas, the W terms represent weight matrices and the b terms represent bias vectors. σ is the logistic sigmoid

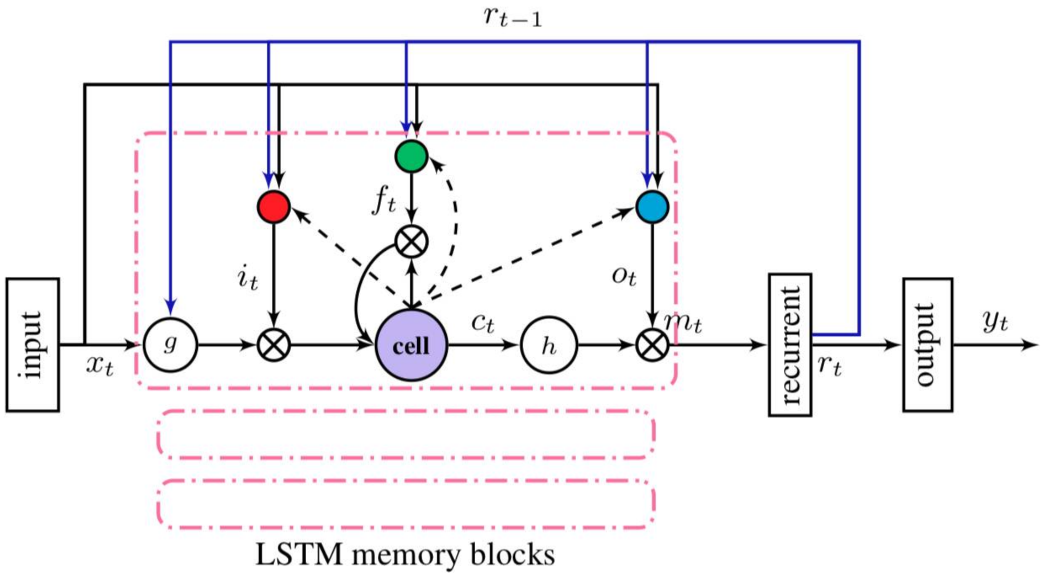


Figure 2.2: a LSTM memory block [4]

function, and g and h are activation functions, generally *tanh*.

In the figure 2.2, there is a input gate, i_t, decides whether the current input should be added to the memory C:

$$i_t = \sigma\left(W_{ix}x_t + W_{im}m_{t-1} + W_{ic}c_{t-1} + b_i\right) \tag{2.8}$$

and an forget gate, f_t, to decide whether to keep the old memory:

$$f_t = \sigma\left(W_{fx}x_t + W_{fm}m_{t-1} + W_{fc}c_{t-1} + b_f\right) \tag{2.9}$$

then update the memory cell c_t:

$$c_t = f_t \odot c_{t-1} + i_t \odot g\left(W_{cx}x_t + W_{cm}m_{t-1} + b_c\right) \tag{2.10}$$

Finally, there is a output gate, o_t, decides whether to output the memory cell at this time-step:

$$o_t = \sigma\left(W_{ox}x_t + W_{om}m_{t-1} + W_{oc}c_t + b_o\right) \tag{2.11}$$

and the final output is m_t:

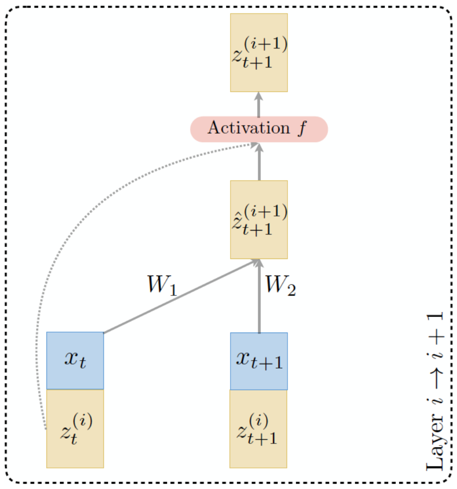
$$m_t = o_t \odot h\left(c_t\right) \tag{2.12}$$

The LSTM structure suggested above is only a basic version. Sak et al. [4] also proposed Deep LSTM Structure and LSTM with Recurrent Projection Layer.

2.2.4 Trellis Network

As mentioned in the previous two sections, the recurrent approach is powerful in solving the sequential tasks. However, the drawback of RNN is also prominent. Since each word in an input sequence needs to be processed sequentially, they are slow to train. Dauphin et al. [17] proposed a Gated Conventional Networks, which can train the words in an input sequence simultaneously. This network immensely improves the training efficiency, and the result of their works proved that the non-recurrent approaches can also compete with strong recurrent models on large scale language tasks.

More advanced than this temporal convolutional networks, Bai et al. [7] present a trellis network, which combined the merits of both Recurrent Networks and convolutional Networks into one novel network named trellis network. From the perspective of model structure, it is a particular kind of temporal convolutional network, though there are two differences: the weights in trellis network are tied across layers, and the input is added to every single layer of the network. Moreover, it generalizes the truncated recurrent networks. A truncated RNN bounds the memory horizon, which can be defined as: for sequence $x_{1:T}$, given an RNN ρ, a corresponding M-truncated RNN ρ^M would calculate the output y_t at time-step t by applying ρ to the sequence $x_{t-M+1:t}$ [7].



(a) TrellisNet at an atomic level

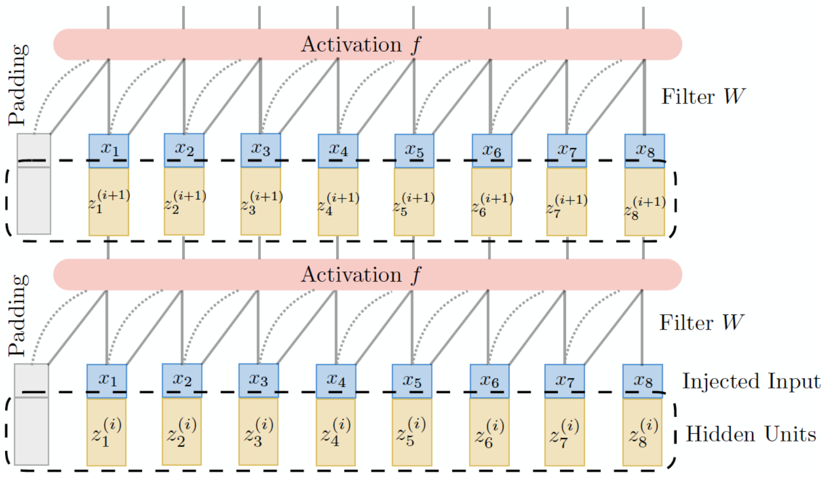

Activation f
Padding
Filter W
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
$z_1^{(i+1)}$ $z_2^{(i+1)}$ $z_3^{(i+1)}$ $z_4^{(i+1)}$ $z_5^{(i+1)}$ $z_6^{(i+1)}$ $z_7^{(i+1)}$ $z_8^{(i+1)}$
Activation f
Padding
Filter W
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
Injected Input
$z_1^{(i)}$ $z_2^{(i)}$ $z_3^{(i)}$ $z_4^{(i)}$ $z_5^{(i)}$ $z_6^{(i)}$ $z_7^{(i)}$ $z_8^{(i)}$
Hidden Units
(b) TrellisNet on a sequence of units

Figure 2.3: The inter-layer of TrellisNet [7]

The basic structure of Trellis Network is described as the figure2.3 (a). A hidden input at time t+1 gathers the information from both the current timestamp t+1, and the previous timestamp t, and each of them consists of two elements - the hidden output of the previous layer i $z^{(i)}$ and an injection of the input vector x; The calculation of the pre-activation output $\hat{z}_{t+1}^{(i+1)} \in \mathbb{R}^r$ follows the linear transformation function:

$$\hat{z}_{t+1}^{(i+1)} = W_1 \begin{bmatrix} x_t \\ z_t^{(i)} \end{bmatrix} + W_2 \begin{bmatrix} x_{t+1} \\ z_{t+1}^{(i)} \end{bmatrix} \tag{2.13}$$

where $W_1, W_2 \in \mathbb{R}^{r\times(p+q)}$ are weights and r is the size of $\hat{z}_{t+1}^{(i+1)}$. The calculation of the output $z_{t+1}^{(i+1)}$ is depend on the pre-activation output $\hat{z}_{t+1}^{(i+1)}$ and the previous layer output $z_t^{(i)}$, through a nonlinear activation function: $f : \mathbb{R}^r \times \mathbb{R}^q \to \mathbb{R}^q$. In this project, they used the LSTM as the activation function.

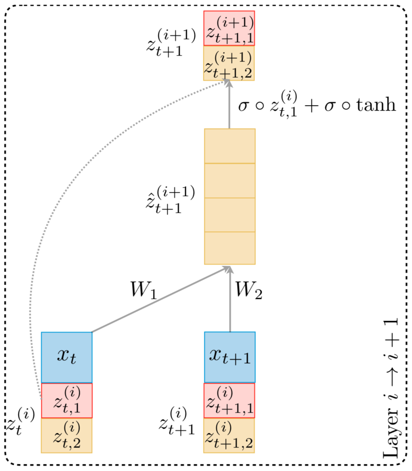


Figure 2.4: An atomic view of LSTM Non-linearity activation function [7]

As figure2.4 shown, the $\hat{z}_{1:T}^{(i+1)}$ is constituted of 4 parts, corresponding to the forget gate, memory cell, input gate and output gate in the LSTM structure respectively. The mathematical expression of $\hat{z}_{1:T}^{(i+1)}$ is:

$$\hat{z}_{1:T}^{(i+1)} = \text{Conv 1D}\left(z_{1:T,2}^{(i)}; W\right) + \tilde{x}_{1:T} = \begin{bmatrix}\hat{z}_{1:T,1} & \hat{z}_{1:T,2} & \hat{z}_{1:T,3} & \hat{z}_{1:T,4}\end{bmatrix}^\top \tag{2.14}$$

The hidden unit is divided into 2 parts: $z_{1:T,1}^{(i+1)}$ and $z_{1:T,1}^{(i+2)}$. The updating of $z_{1:T,1}^{(i+1)}$ is similar to the updating the memory cell in LSTM, following the function:

$$z_{1:T,1}^{(i+1)} = \sigma\left(\hat{z}_{1:T,1}\right) \circ z_{0:T-1,1}^{(i)} + \sigma\left(\hat{z}_{1:T,2}\right) \circ \tanh\left(\hat{z}_{1:T,3}\right) \tag{2.15}$$

and the $z_{1:T,2}^{(i+1)}$ is analogous to the LSTM output, and it is updated by:

$$z_{1:T,2}^{(i+1)} = \sigma\left(\hat{z}_{1:T,4}\right) \circ \tanh\left(z_{1:T,1}^{(i+1)}\right) \tag{2.16}$$

The whole network consists of many single atomic units across time and depth.

By connecting many single atomic trellis nets, the whole TrellisNet can be constructed. With the sequence $x_{1:T}$, the weights of the all layers and timestamps are tight (i.e., the weights are the same). Thus, to improve the computational efficiency, the linear transformation $\tilde{x}_{t+1} = W_1^x x_t + W_2^x x_{t+1}$ was precomputed, and was applied to all layers, combining with the hidden unit $W_1^z z_t^{(i)} + W_2^z z_{t+1}^{(i)}$ [7].

The result of Bai et al.'s work [7] suggests that in terms of language model, the performance of TrellisNet outweighs many previous models. Specifically, for the word-level language modelling on the WikiText-103 (WT103) dataset, the test perplexity of TrellisNet is 35.4% lower than Generic TCN (Bai et al. [18]), 11.5% lower than AWD-QRNN (Merity et al. [19]) and 7.6% lower than Relational Memory Core (Sabtoro et al. [20]). Trellis Network is expected to construct a deeper network structure in sequential model.

2.3 Optimization

Even with a well-designed skeleton, the model can suffer from some problems in actual application, such as overfitting problem and the computation complexity problem. This section will introduce some method to solve these problems.

2.3.1 Dropout

- *Variational Dropout*
  RNNs are effective models that perform well in sequential tasks, however, it can suffer from overfitting problems. Dropout is the most commonly-used regularization technique to solve overfitting problems. In many machine learning models, dropout was only applied to input and output. Nevertheless, according to Zaremba et al. [21], the traditional dropout was inefficient on RNN, and it was considered to bring models with the noise that would be amplified in a long sequence. Gal and Ghahramani [22] provided a way to understanding common deep learning skills from a Bayesian perspective, and they stated that dropout can be interpreted as a posterior variant approximation of Bayesian neural networks. By apply approximate variational inference in common RNN models, Gal and Ghahramani [23] proposed a new dropout scheme. They used same dropout mask for each diagonal of the network, and the algorithm is shown in the figure 2.5 (Different dropout masks are distinguished by colour; dashed lines means no dropout on the network connection). Their works empirically proved that this new dropout technique is efficient on language model tasks.

- *Recurrent Weight Dropout*
  In terms of Language Model, the regularizing on hidden-to-hidden weights can be very helpful to avoid overfitting problems among recurrent connections, and this was proved by Merity et al.'s work [24]. They implemented a weighted-dropped LSTM and the scheme of their work may also be referred to the other sequence model.

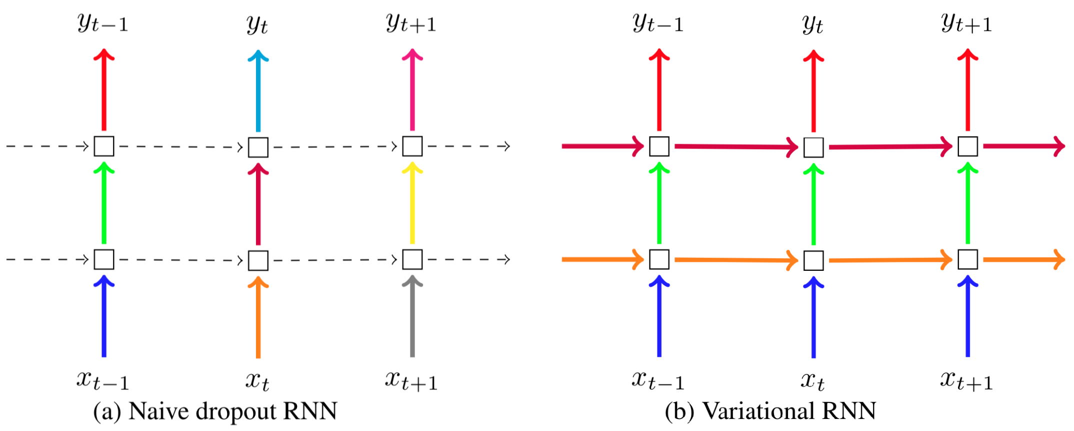


Figure 2.5: Variational Dropout [23]

2.3.2 Deep Supervision

In a training process of deep network, many models only track the loss of final output from the last layer. However, the features included in the hidden layers can also play a significant role. Xie and Tu [25] clarified the necessity of the deep supervision. They stated that each network layer can work as a singleton network, having effects on the overall performance of the whole model. Lee et al.'s work [26] tried to apply "compassion" loss function to each intermediate layer. Their work showed that this deep supervision acts like a string regularization, and helps to avoid detrimental overfitting.

2.3.3 Weight Normalization

Salimans and Kingma [27] present a reparameterization method, weight normalisation to accelerate the convergence of SGD, and it is realized by applying the formula below:

$$\mathbf{w} = \frac{g}{\|\mathbf{v}\|}\mathbf{v} \tag{2.17}$$

where g is a scalar, v is a vector and $\|\mathbf{v}\|$ is the norm of vector v. Thus, in this formulation, g measures the size of w and $\frac{v}{\|\mathbf{v}\|}$ represents the direction. With weight normalisation, the formula to calculate gradient is:

$$\nabla_g L = \frac{\nabla_{\mathbf{w}} L \cdot \mathbf{v}}{\|\mathbf{v}\|}, \quad \nabla_{\mathbf{v}} L = \frac{g}{\|\mathbf{v}\|}\nabla_{\mathbf{w}} L - \frac{g\nabla_g L}{\|\mathbf{v}\|^2}\mathbf{v} \tag{2.18}$$

or alternatively:

$$\nabla_{\mathbf{v}} L = \frac{g}{\|\mathbf{v}\|} M_{\mathbf{w}} \nabla_{\mathbf{w}} L, \quad \text{with} \quad \mathrm{M}_{\mathbf{w}} = \mathrm{I} - \frac{\mathbf{w}\mathbf{w}'}{\|\mathbf{w}\|^2} \tag{2.19}$$

With the weight normalization, the learning rate can be set higher, leading to a faster training process.

2.3.4 Mixture of Softmax

In language modelling, the probability of the next word can be calculated by the softmax function. However, the number of potential candidates is generally large. Regarding the language model as a matrix factorization problem, the factorized matrix can be high-rank. Yang et al. [28] argue that this ordinary softmax method has insufficient ability to help model this kind of tasks. Thus, they proposed Mixture of Softmax to solve this Softmax Bottleneck. MoS has better expressiveness for an embedding dimension. The structure of Mixture of Softmax is:

$$P_\theta(x|c) = \sum_{k=1}^{K} \pi_{c,k} \frac{\exp \mathbf{h}_{c,k}^\top \mathbf{w}_x}{\sum_{x'} \exp \mathbf{h}_{c,k}^\top \mathbf{w}_{x'}}; \text{ s.t. } \sum_{k=1}^{K} \pi_{c,k} = 1 \tag{2.20}$$

$$\pi_{c_t,k} = \frac{\exp \mathbf{w}_{\pi,k}^\top \mathbf{g}_t}{\sum_{k'=1}^{K} \exp \mathbf{w}_{\pi,k'}^\top \mathbf{g}_t} \tag{2.21}$$

$$\mathbf{h}_{c_t,k} = \tanh\left(\mathbf{W}_{h,k}\mathbf{g}_t\right) \tag{2.22}$$

$$\tag{2.23}$$

where $\pi_{c,k}$ is the weight of park k, and $\mathbf{h}_{c,k}^\top$ is the represents the k^{th} context vector that associated with context C. This is equivalent to calculating K softmax, and then weighted summation.

2.4 Evaluation Method

To measure the performance of models, various evaluation criteria can be applied. This section will introduce the evaluation methods used in this project.

2.4.1 Normalized Discounted Cumulative Gain

To evaluate the quality of the rank algorithm, Normalized Discounted Cumulative Gain (NDCG) can be an important evaluation index. Discounted Cumulative Gain (DCG) [29] [30] measures if the relevant documents are ranked before non-relevant documents:

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{2.24}$$

where rel_i is the relevance score for the document at position i. In this formula, $2^{rel_i} - 1$ represent the rank gain, and $\log_2(i+1)$ represent the position discount. To evaluate ranks results in a consistent way, Wang et al. [31] proposed Normalized DCG (NDCG):

$$\mathrm{nDCG_p} = \frac{\mathrm{DCG_p}}{\mathrm{IDCG_p}} \tag{2.25}$$

where IDCG is the DCG in the best case.

Chapter 3

Data Analysis

This chapter will introduce the training data and test data used in this project, and will analyse the data to give some suggestion for project implementation.

3.1 Data Description

The data used in this program are all from Spotifys public dataset of playlists, and they are accessible at https://www.crowdai.org/challenges/spotify-sequential-skip-prediction-challenge. As the project is focused on model design, the minimally sized version of dataset was preferentially applied, and the whole dataset is also available online for further training.

The dataset mainly consists of two parts: the dataset for track features and the dataset recording session logs. In the the dataset for track features, every track has a unique 'track id'. The data available for each track describes its music characters such as: music duration, release year, popularity, music style and acoustic features. There are 29 features in total for each track, and the detail of all the features included can be found in the Spotify dataset website. The session log dataset records the track sequences in each session, and for each track that user played, there is some metadata describing how user response to that track. This metadata gives the information include the skip information (when the user skips the track), the switch indicator to show if the user changed context between this track and the last track, the seek indicator tells the number of times that the user seek forward or backward within a track, the premium indicator to show if user is on premium and so on. The 'session id' and session position can be cooperated to distinguish different sessions in a specific timestamp, and the session logs dataset can connect with the track feature dataset by the variable 'track id'. The detail of all the available session characters mentioned above and beyond above can be found on Spotify website as well.

It is worth to note that there is no user data available. In the session logs dataset, different sessions might be created by different users. Hence, it is impractical to design a user model.

3.2 Exploratory Data Analysis

The Exploratory Data Analysis (EDA) provides statistical outputs and offers some suggestions for project experiments.

- *The General Data Analysis*

  In the dataset used by this project, there are 50704 tracks, 10000 sessions and 167880 session logs in total. The session length of each session ranges from 10 to 20 (the number of tracks played in a single session is ranging from 10 to 20), and 50.7% sessions have the length of 20. Thus, if the project would like to apply a mini-batch approach, it is applicable to pad each sequence to the length of 20 (The visualization of the session length distribution is appended to A.1). In addition, the play volume contributed by Spotify premiums accounts for 81.0% of the total play volume, however, the data analysis also shows that a premium and a non-premium have approximately the same probability to skip a track. Therefore, it seems that there is no need to distinguish premiums and non-premiums while training. Furthermore, among all the session logs, 59029 tracks out of 167880 were not skipped by users, 70762 were skipped at beginning, and only 38089 were partially skipped, which may indicate that without data sampling, the model might be hard to learn the music features of the track that were partially skipped (A.2).

- *The Correlation Analysis*

  The correlation analysis of track features dataset shows that even though there are 29 features for each track, some features are strongly correlated. For instance, a track with strong beat strength tends to have a relatively high score on bounciness, danceability and dyn range mean as well. The heat map for correlations between some music feature has been appended to A.3 at the end of this report. This result can give some guidance once the project need to reduce the dimension of training data. As the correlation analysis for session logs dataset, the results show that whether a user skips a track is not strongly tied with whether he switches a content, or whether he seeks forward or backward that track.

- *The Distribution Analysis*

  Another analysis of track feature distribution found that in the music library, 73.6% of tracks are released later than 2010 (while the release year of all tracks ranges from 1950 to 2018); 93.1% tracks' US popularity estimate is more than 98 (while the popularity estimate ranges from 90.02 to 100.00 for all the tracks in the library); and 89.78% of tracks have the time signature of 4 (while there are 5 potential candidate value for all tracks, which are 0, 1, 2, 3, 4, 5). This result shows that these track features distributions are unbalanced, and thus the tracks with minority features might be less played by users. By analysing the session logs, it found that for the tracks played in the whole session logs, 85.5% are released later than 2010; 96.4% tracks' US popularity estimate is more than 98 and 93.6% have the time signature of 4. By directly applying these session logs to the model training process, it might cause a disruption to the final prediction result.

  There are some insightful discoveries produced by combining track features dataset and sessions log dataset together. To evaluate if users have a preference on a specific popularity of tracks, this project visualised the standard deviation of tracks' 'US popularity estimate' within each session, the result is shown in 3.1.

  As this Figure shows, most of the standard deviation of popularity estimate is less than 0.5, while the largest is 4.9, which indicates that users tend to listen to the music of the same level of popularity. To avoid the influence of the unbalanced distribution on 'us popularity

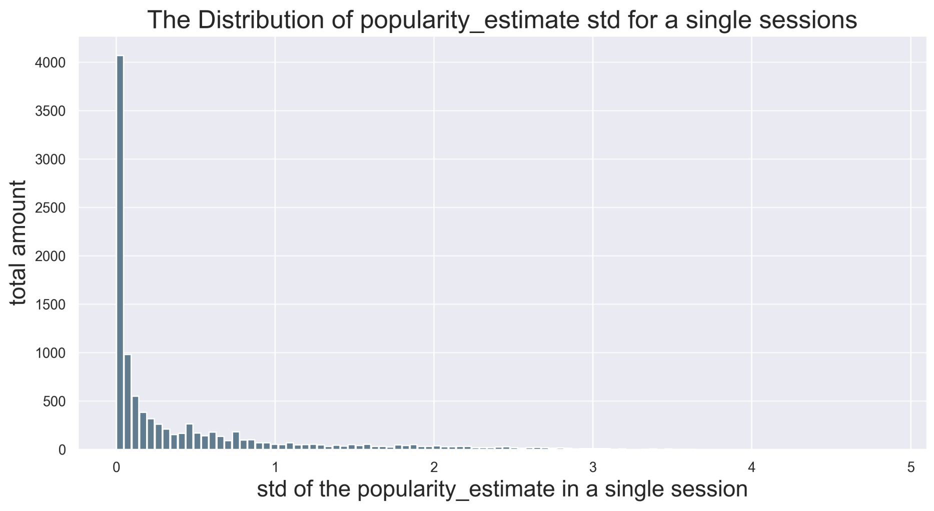


Figure 3.1: Std of Popularity for all tracks

estimate', the sessions with only the tracks that have 'us popularity estimate' more than 98 are excluded from the dataset. Consequently, the dataset now only include the sessions that have at least one unpopular track. The standard deviation visualisation for this new dataset is shown in 3.2.

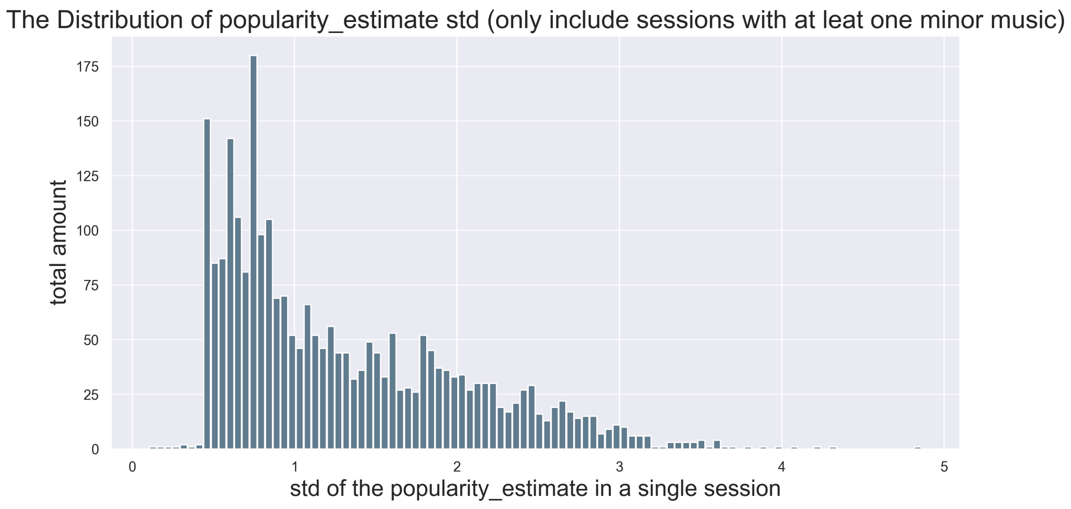


Figure 3.2: Std of Popularity for minor tracks

This figure illustrates that most of the sessions still tend to have a relatively small standard deviation, i.e., while some users prefer pop songs, the others are fond of minority songs. Consequently, it can be concluded that 'us popularity estimate' is an important index, however, as it showed previously, the distribution of 'us popularity estimate' is unbalanced. Therefore, it is necessary to do a down sampling or up sampling for the original data. The same importance showed in the release yeas of tracks (the distribution appended in A.1 A.2).

Some users like modern songs, while others like retro songs. Unfortunately, the distribution of ‘release year’ in the training dataset is not balanced either. Therefore, it is unwise to straightly use this dataset for training.

Chapter 4

Methodology and Experiment

This chapter will introduce the machine learning models, technical implementation details and evaluation methods for the tasks mentioned in Chapter 1.3.

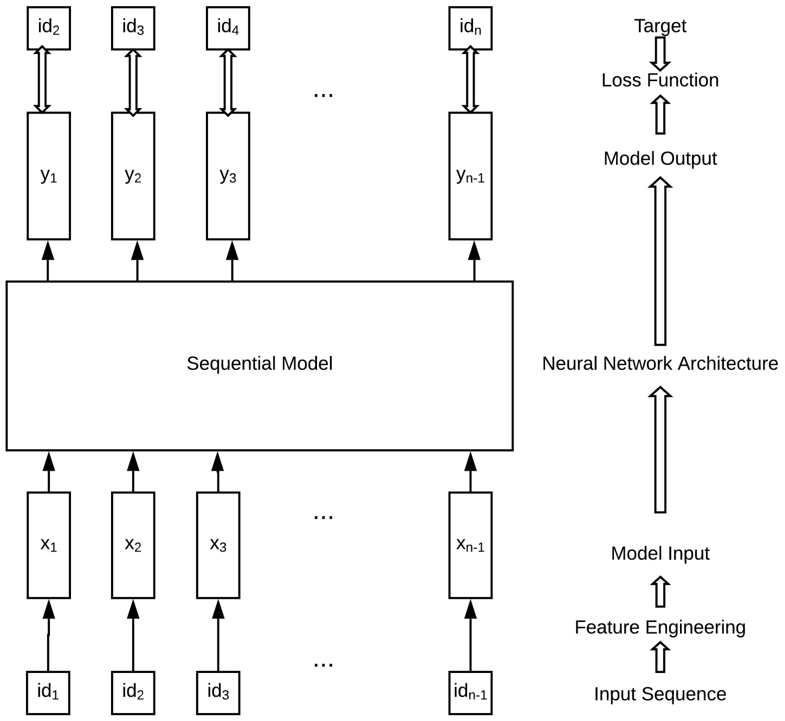


Figure 4.1: The Structure of the training process

The structure of the training process is illustrated in figure 4.1, and this chapter will be carried forward following this structure.

4.1 Feature Engineering

As suggested in 3.2, each track is described by 29 features, which compromise the music duration, release year, popularity, music style and so on. Different tracks are distinguished by the different combination of these 29 features, and users' music preference can be reflected in the features of tracks they played. Although it is easy for a human to identify a track by its track id, machines require more specific data representation. To inform models the difference among different tracks, the feature engineering was applied to convert the track features to a mathematical expression that is understandable to machines. The discrete feature variables were converted into integers where each integer represents a class of feature values. For continuous variables, they were adjusted to form a distribution with mean equaling to 0 and standard deviation equaling to 1. Finally, each feature is represented by a scalar, and the features of each track is converted to a 29-dimensional vector. Similar to the word embedding [32], which shows the distance between two words, the distance between music embeddings can also reflect the similarity between two tracks.

4.2 Approaches to Applying Skip Information to Model Input

As explained in the 1.3, this project aims to figure out how to apply language model to the music recommendation tasks. The language model has been successfully applied in many Natural Language Processing (NLP) Tasks, such as sentence generation, machine translation, speech recognition. By analogy with language model predicting the probability distribution of the following words, this project tried to take the advantage of the track sequence in each session and predict the most probable next track that users might like. However, there is a big difference between NLP problem and session-based music recommendation. In NLP, a sentence is created with a convention, and the sequence of words tends to be logical. In terms of music recommendation, the tracks sequence is sometimes created passively, and some of them created by the music service provider. Hence, there is no strict logic existed in the tracks sequences, as some of these tracks are not attractive to users. Fortunately, whether user like a track could be revealed by the skip information, i.e., if a user enjoy a track, he would stay in this track and not skip it. By contrast, once the track appeared in a session is disliked by that user, he would probably skip this track directly. Thus, making good use of this information can help the sequence become more logical.

4.2.1 Approach 1: Deleting Skipped Tracks

The first approach tried in this project is to delete the skipped tracks directly. This is achieved by using the skip information as a condition of judgment (0 means not skip any part of that track, 1 means skip a small part, 2 means skip half and 3 means skip the whole track). When the skip information is more than 1, this project regards this track as the one that was skipped by user, so this track this track will not be put into the model for training. The visualisation of this process is shown in 4.2

The logic behind this approach is that the final model input only includes the tracks that this user like, which means all the features included in the training data are preferred by user, thus the

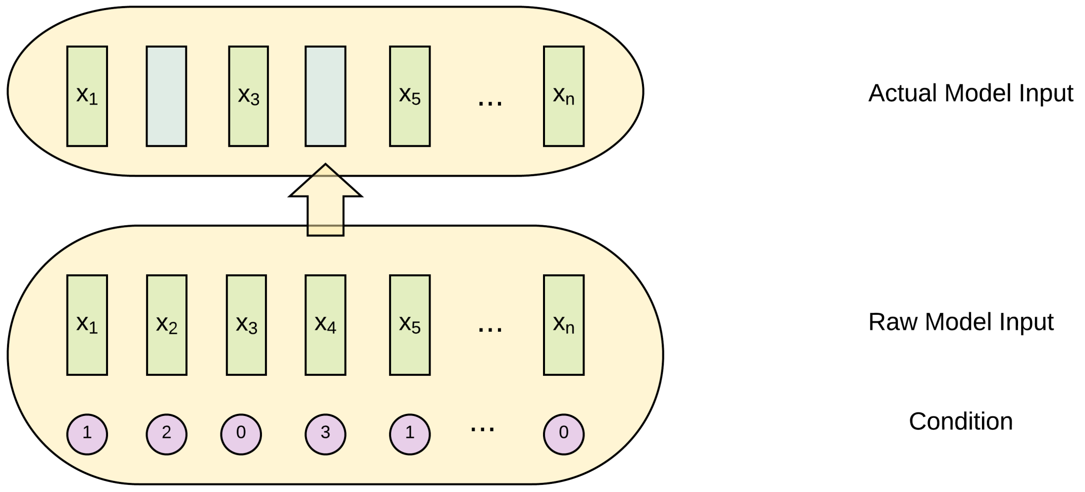


Figure 4.2: Deleting Skipped Tracks Approach

next predicted output supposed to be liked by user as well. The merit of this approach might be that it has a strong filtering signal for liked tracks, however, this approach prevents deep learning models from learning the features that users dislike.

4.2.2 Approach 2: Adding Session Information

Another approach tried in this project is to concat the session information to the music embedding vectors. Recall that in Chapter 3.1, it mentioned that there is another dataset for session logs, which records how user response to a track in a session. The skip information is included in the session information. By adding the skip information to the input vector, the model is able to identify users' preferences in theory. In addition to skip information, the session information comprises some other logs information such as the switch indicator to show if the user changed context between this track and the last track, and the premium indicator to show if the user was on premium. In total, there are 16 factors (include skip information) for a single session, and there are converted to a 16-dimensional vector. By concating this 16-dimensional vector to the track feature embedding, the final model input is formed (This is visualised in 4.3).

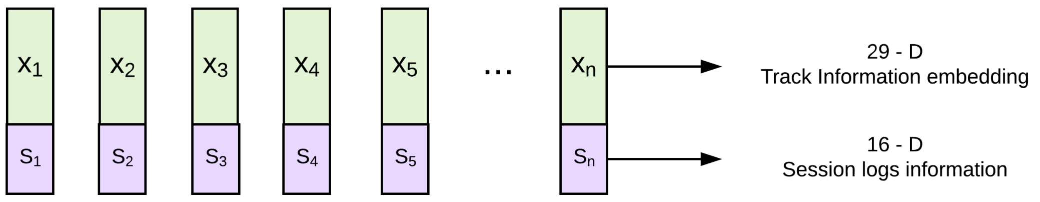


Figure 4.3: Adding Session Information Approach

The merit of this approach might be that it provides more user behaviour data, so that model might predict more accurately based on these additional data. More importantly, the model now is able to learn the group of track features that is not liked by users. However, this summary of information may cause the ambiguity in the hidden state of a neural network.

4.3 Neural Network Architecture

4.3.1 Published Deep Learning Algorithm

This project aimed to apply language model to the music sequence tasks, and some successful neural network architecture in NLP tasks will be refered to. Bai et al. [7] proposed a trellis network, which improved performance in perplexity than many other recent neural network architecture. Therefore, this project tried to figure out if whether this network would be efficient on music sequence as well. In addition, the traditional Recurrent Neural Network (traditional RNN) and Long Short-Term Memory (LSTM) will be taken as baseline models.

4.3.2 Optimization

To boost the training efficiency and further improve the model performance on the future unseen data, some techniques were applied and this section will introduce them in detail.

- *Parameter Tuning*
  There are many parameters in each algorithm, and a proper combination of parameters' values would lead to a more accurate model result than others. Parameter Tuning aims to find out the optimal parameter values combination. A specific value of a single parameter may lead to a better model result than other values, however, this value may not work well with the other parameters in the same model. Therefore, this project applied grid search to figure out the optimal value combination for each parameter, aiming to get the optimal model result as a whole.

- *Variational Dropout and Recurrent weight dropout*
  The overfitting problem is a big difficulty for many machine learning model. With the increase of training times, the model tends to be increasing more fitted with the training data, and perform increasingly better on training data. However, it might lead to a worse result on future unseen data. That is the overfitting problem. To solve the overfitting problem, the model can use 'early stop' technique to reduce training times, however, this is hard to control. This project used dropout to avoid overfitting problems. The variational dropout [23] and recurrent weight dropout [24] was applied to the trellis network. Slightly different from Gal and Ghahramani's work [23], the variational dropout in trellis network model used the same mask in both vertical and horizontal direction.

- *Other techniques to optimize Trellis Network*
  The trellis network injected loss functions to hidden layers for deep supervision [7]. and the final loss combined the auxiliary loss from the hidden layers with the original loss from the

last output layer:

$$L_{\text{total}} = L_{\text{orig}} + \lambda \cdot L_{\text{aux}} \tag{4.1}$$

where λ is the weight of the auxiliary loss.The weight normalisation [27] was also applied to the trellis network. By applying weight normalisation, the learning rate can be set higher, thus the training time will be shorter. In addition, the trellis model used the Mixture of Softmax [28] instead of the traditional Softmax function for method Combination 1 and Combination 3 (4.4), which will be explained in the next section.

4.4 Output and Loss Function

This project attempted two different output methods. In the NLP problems, it is common for a model to output a vector with a dimension equalling to the length of the whole corpus, and each scalar in that vector corresponds to a word in the corpus. After calculating the cross entropy with the next words and doing backpropagation, the probability of being the next words for all the valid words can be shown in the output vector. Similarly, in the first output method of this project, the dimension of the decoder output was set to be the length of music library, the next track was taken as the training target and the cross entropy function was taken as the loss function (visualized in 4.4(a)).

This project also tried to output a 29-dimensional vector, using mean squared error (MSEloss function) to directly compare the predicted features and the actual features of the next track (the actual next track, which is acted as the training target, would be embedded to a 29-dimensional vector in this case). This method is expected to reduce the training time, and it might lead to a more accurate result (visualized in 4.4(b)).

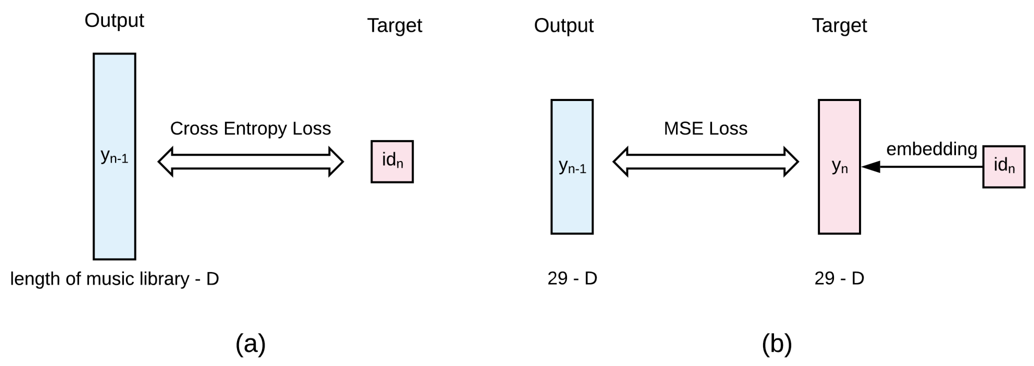


Figure 4.4: The output of Neural networks

To summarise the methods to deal with the model input and output, there 2 approaches for preparing inputs and 2 approaches for exporting outputs, lead to 4 combinations in total. They are Combination 1: input with non-skipped samples (4.2.1) + output of music library length;

Combination 2: input with non-skipped samples (4.2.1)+ output of track features length; Combination 3: input with the whole samples (4.2.2) + output of music library length; Combination 4: input with the whole samples (4.2.2) + output of track features length. It is worth to note here, in Combination3 and Combination4, all the session tracks, including the liked tracks and disliked tracks, were used in training. Therefore, in language model, when the next track id_{n+1} is taken as the training target of the current state id_n, it is unknown whether id_{n+1} is a positive target. Since our aim is to predict the next song that would be liked by user, the actual disliked target can bring confusion to neural network models. Therefore, for Combination 3, the disliked targets were excluded from propagation (This is different from Combination 1, because all the tracks in Combination 3 were participated in training process while the only the non-skipped tracks in Combination 1 were included). For combination 4, one more scalar, representing the skip information, was added to the output as well as the target embedded vector, so that the actual dimension of the output in Combination 3 is 30. With this additional scalar, model should know how much the user likes the next song (the training target), and the model output now includes information of the predicted features of the next song and how much user might like these features.

4.5 Evaluation

Recall that in this project, the task is to help user to rank the songs that haven't been played in their playlist to cater to their preference. To be specific for this project, there is a sequence:

$$S^i_{i:T} = t^i_1, t^i_2, ..., t^i_{T/2}, t^i_{T/2+1}, ..., t^i_T \tag{4.2}$$

Suppose the sequence $t^i_1, t^i_2, ..., t^i_{T/2}$ has been played by users, the task of this project is to rank the sequence $t^i_{T/2+1}, ..., t^i_T$, so that user is expected to prefer top-placed songs more than bottom-placed songs.

As the task of this project is ranking, the performance on ranking result should be the main criteria to decide whether the model is well designed. This section will explain how this project use the model output to rank the unplayed songs, and how to evaluate the ranking result.

4.5.1 Ranking Method

Once the model has been fully trained, they can be applied to ranking tasks. In the ranking processes, it supposes that we know the sequence of the first half of a session: $t^i_1, t^i_2, ..., t^i_{T/2}$, and we know the user response to each of these tracks. Then, by putting this sequence and user response to those trained models, they will output a vector, which can be used to predict the most probable next track that is expected to be liked by users.

In model Combination 1 and Combination 3, the model will output a vector with a dimension equalling to the length of the whole music library. The probability of being the next loved track for each track in the library is represented by a single element in that output vector. Therefore, to rank the $t^i_{T/2+1}, ..., t^i_T$, the aim is to find the position of these candidates in that vector, and get the score of being the next track for that candidate. The final order of ranking is sorted according to these scores getting from the output vector.

In model Combination 2 and Combination 4, the model will output a 29-dimensional vector, of which each element in that vector corresponds to a feature of the predicted track. Knowing the track id of each ranking candidates, we embedded these n candidates to n 29-dimensional vectors. Comparing the distance between the predicted track features and these candidate track features respectively, it is feasible to find out the most similar track. This distance was measured by cosine similarity, and the candidate with the highest cosine similarity score was considered to be the most similar song to the predicted song, and thus was placed at the first place. It is worth to note here, as mentioned in Chapter 4.4, for Combination 4, there is a small modification, which concats one more scalar to the output vector, changing the output dimension to 30. Therefore, when ranking the candidates, it should add one more scalar to the candidate embedding as well. As this track was expected to be liked by users, this scalar added should show a positive signal (in this project, this scalar added was set to be 0, representing this track was expected to be not skipped).

4.5.2 Ranking Evaluation Metric

There are many evaluation metrics for ranking algorithms, the metric applied in this project is Normalized Discounted cumulative gain (NDCG). The NDCG is calculated using the following formula. The DCG should be firstly calculated:

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{4.3}$$

where rel_i is the relevance score for document at position i. In this formula, $2^{rel_i} - 1$ represent the rank gain, and $\log_2(i + 1)$ represent the position discount. To evaluate ranks results in a consistent way,the NDCG is calculated by:

$$\mathrm{nDCG_p} = \frac{\mathrm{DCG_p}}{\mathrm{IDCG_p}} \tag{4.4}$$

where IDCG is DCG in the best case. In the first evaluation step of this project, the NDCG score measures the performance of the whole sessions in the testing dataset. Then, to discover if the models would give different results on sessions with different skip proportion, this project divide the session into two groups, one group with the skip proportion of the first half of session more than x, and another group with the skip proportion of the first half of session less than x, and then test the NDCG score on them respectively. There were three values set to x: 25%, 50% and 75%.

4.6 Experiment Set Up

This whole project was implemented by Python3. Pandas and Numpy were used to handle the data and PyTorch was used to process some machine learning models more efficiently. The optimizer used in training is stochastic gradient descent (SGD) [33].

In this project, the data was divided into the training data, validation data and test data. The training data was used to train the model, the validation data was used to examine the training result on the unseen data, and decide which group of parameters to save. The training time for most of the models implemented in this project is remarkably long. Hence, to reduce the training time, the mini-batch SGD was applied. In the dataset used in this project, the sequence length of each session varies from 10 to 20. To make the data trainable for mini-batch, the sessions less than 20 were padding to the length of 20. However, the padding elements were excluded from the backpropagation so they have no influence on the model parameter updating. In the ranking process, as the whole length of a session is 20, and the length of the first half of session is 10, the x is set to be 3,5,8 when testing the model result on different groups of sessions with different skip proportion 4.5.2.

Chapter 5

Results and Discussion

This chapter examines the three models mentioned in the last chapters, using different methods to process the input and output. This chapter will show all the experiments results, and discuss the insights got from these results.

5.1 Testing on the whole test data

The NDCG was used to examine the ranking performance of different models. NDCG is a normalised DCG, so its maximal value is 1. The test depended on the test data, using the information given by the first 10 tracks to rank the rest of the tracks in the same session. The ranking result is evaluated by the skip information the rest of tracks. Without any models, the worst NDCG score for the test data is 0.66933, and the NDCG score for ranking by tracks popularity is 0.80901. These two values were taken as baseline values.

The table 5.1 shows the testing results on the while test dataset. Here is a recall for the meaning of each combination in the Input and Output Method column:
Combination 1: input with non-skipped samples (4.2.1) + output of music library length (i.e., input approach 1 + output method 1);
Combination 2: input with non-skipped samples (4.2.1)+ output of track features length (i.e., input approach 1 + output method 2);
Combination 3: input with the whole samples (4.2.2) + output of music library length (i.e., input approach 2 + output method 1);
Combination 4: input with the whole samples (4.2.2) + output of track features length (i.e., input

Input and output Combination	RNN	LSTM	Trellis
Combination 1	0.80736	0.81110	0.81129
Combination 2	0.80317	0.80975	0.81082
Combination 3	0.81625	0.82153	0.82529
Combination 4	0.80892	0.81297	0.81362

Table 5.1: The NDCG Score on the Whole Test Dataset

approach 2 + output method 2).

As this table 5.1 illustrates, all of them improved the model from the worst case (0.66933). The best result (Trellis Model using combination 3), improves the worst case by 0.156 and improves the popularity-based algorithm by 0.016. Nevertheless, it seems that the ranking abilities of some algorithms are inferior to the popularity-based ranking, which makes these algorithms meaningless.

To compare two methods of processing input, the approach 2, which concats the session information (includes skip information) to the track feature vectors, performs better than approach 1, which directly excludes the skipped tracks from training process. Each of the NDCG score on approach 2 is higher than that on approach 1 with the same network model and output method. This might be because the approach two grasps more session information than approach 1. Since it knows when user paused the track and whether user sought forward, it can give more accurate prediction results. This guess is concluded from the test result on the whole dataset only, the analysis of test results on different groups of test data will give more insights 5.2.

Comparing two output methods, it seems that the result is similar in Combination 1 and Combination 2, even though the music library decoder in the method 1 performance a slightly better, with the NDCG score 0.0042, 0.0015 and 0.0005 higher than method two on RNN, LSTM and Trellis network respectively.Before the experiments, the second output method was expected to perform better, because it seems to have more direct training to the features of tracks in the training dataset. However, the result proved that by output the long vector, the features of tracks can also be trained adequately. As for combination 3 and combination 4, output method 1 still perform better than method 2, but with a greater difference. The NDCG score on Combination 3 is 0.0073, 0.0086 and 0.0117 higher than that on Combination 4 for RNN, LSTM and Trellis network respectively. This bigger difference may come from the different method to deal with the skipped target as mentioned in Chapter 4.4. The additional scalar added to the output vectors in Combination 4 was meant to help model distinguish the liked target and disliked target, however, it might lead to the ambiguity problem. It might be difficult for models to distinguish the boundaries between signals and features.

The results among three neural network models show that Trellis outperform the other two, while LSTM has very close NDCG scores to the Trellis network. On the music sequence tasks, he model ability of RNN architecture is shown to be much weaker than the other two. That might due to the limitation of the traditional RNN structure. Compared with LSTM, it can be hard for the traditional RNN to capture the features from a long distance, and RNN is easy to be overfitted. LSTM architecture and Trellis Architecture are less limited by these problems, and offer an acceptable result for music ranking tasks. Trellis Architecture, especially, provided a more prominent performance. In addition, with the same degree of network complexity, the training time for Trellis network is shorter than LSTM, therefore, with limited time, the Trellis network is allowed to train a deeper network. Bai et al. [7] have proved that the Trellis network performance well on the language model in NLP tasks, and this project shows that the Trellis network can be efficient on the music sequence tasks as well.

Input and output Combination	Value of x	RNN	LSTM	Trellis
Combination 1	3	0.90716	0.90886	0.91277
	5	0.89457	0.89427	0.89331
	8	0.84576	0.84123	0.85458
Combination 2	3	0.89726	0.90012	0.89684
	5	0.89417	0.89082	0.89130
	8	0.83692	0.84709	0.85034
Combination 3	3	0.88944	0.88414	0.88948
	5	0.85846	0.85476	0.86099
	8	0.83043	0.83359	0.83687
Combination 4	3	0.90333	0.90212	0.90403
	5	0.86248	0.87467	0.864101
	8	0.82308	0.82821	0.82363

Table 5.2: The NDCG score on the sessions where the number of skipped tracks is LESS than x

Input and output Combination	Value of x	RNN	LSTM	Trellis
Combination 1	3	0.77920	0.78350	0.78265
	5	0.74228	0.74902	0.75008
	8	0.68948	0.71858	0.67841
Combination 2	3	0.77662	0.78425	0.78654
	5	0.73526	0.74925	0.75075
	8	0.69958	0.69513	0.69009
Combination 3	3	0.79219	0.80094	0.80417
	5	0.77999	0.79298	0.79462
	8	0.74539	0.76122	0.76735
Combination 4	3	0.78034	0.78365	0.78389
	5	0.76428	0.75995	0.77023
	8	0.72291	0.73673	0.76357

Table 5.3: The NDCG score on the sessions where the number of skipped tracks is MORE than x

5.2 Testing on different groups of test data

The previous section showed the test result on the whole test dataset. However, this project makes a hypothesis that the skip information on the test data can lead to different model performance, i.e., in a single session, the model can rank the unplayed tracks in a more accurate order if the played songs are less skipped by users. Therefore, this section will divide the test data into two groups: the Group 1 in where there are less than x tracks were skipped by users among the first 10 tracks in a session, and the Group 2 where there are less than x tracks were skipped.

The test result for the group 1 is shown in the table 5.2 and the result for group 2 is shown in the table 5.3. With this result, it can provide more information about the merits of each model and methods implemented in this project.

Firstly, the overall comparisons among the three tables, 5.1, 5.2 and 5.3 show that the session with less skipped tracks in the first 10 played tracks can provide ranking algorithms with more valid information, so that the model can rank the unplayed tracks closer to user's preference. The overall performance on table 5.2 is better than that on table 5.1, and the overall performance on table 5.1 is better than that on table 5.3. Additionally, in the table 5.2 and 5.3, the NDCG scores is reduced along with the rises in the number of skipped tracks in the first half of session. This information can indicate that the models implemented in this project requires adequate non-skipped tracks to give an acceptable ranking result. If a user skipped most of the tracks that he played, it would be hard for models to infer his music preference, and thus cannot offer a satisfactory ranking result. When user skipped most of the tracks, the neural networks implemented in this project even worked worse than the popularity-based algorithm (the NDCG scores in table 5.3 is all less than popularity-based ranking result, 0.80901). Thus, from this result, it can recommend that the system can use the popularity-based algorithm to rank rather than the neural network implemented in this project. When there are enough non-skipped tracks, the input method 1, which directly ignores the skipped tracks during training, performs better than the input method 2, which includes all the tracks in the training process. With the same value of x and neural network, all the NDCG scores in Combination 1 are higher than Combination 3, and most of the NDCG scores on the Combination 2 are higher than Combination 4. Even though the input methods two learned more session information, method 1 gives much stronger signal on for tracks that user like. The input method 2, however, can be harder to learn the preference included in the vectors. The merits of input method 2 were shown in the case where there are no enough non-skipped tracks. In the table 5.3, with the same value of x and neural network, the NDCG scores in Combination 2 and 4 are higher than Combination 1 and 3, and these differences are increasingly prominent along with the increase of the value x. With limited non-skipped tracks, method 1 have no enough resource to learn user preference, and it do not learn the negative samples at all. By contrast, the method 2 learns the features of skipped tracks, helping infer user's preference to some extent. However, as the result is worse than the popularity-based rank, this method is meaningless.

For combination 1 and 2, the output method comparison is similar to that in Chapter 5.1. With the same value of x and neural network, the NDCG scores in combination 1 and 2 are very close. It is interesting to find that when there are enough non-skipped tracks, the output method 2 outperforms output method 1 (table 5.2). By contrast, the output method 1 outperforms output method 2 in the table 5.3. This might be because when there are few skipped tracks (in table5.2), the Combination 4 is less disturbed by negative samples during the ranking process.

In terms of the neural network, it found that the advantages of LSTM and Trellis network are mainly shown in case of a large proportion of skip, and they have a very slight superiority with many non-skipped tracks. This demonstrates that the traditional RNN have the ability to learn simple and controllable information, however, for more complex and irregular information, LSTM and trellis network are more competent.

5.3 Revelation on new ranking method

Learned from the findings of the testing results above, this project tried to improve the algorithm for ranking on the whole test dataset. For the two input method, since method 1 has more direct training on the songs liked by users, it provides a more prominent performance on the sessions

Input and output Combination	RNN	LSTM	Trellis
Combination 1	0.80809	0.79770	0.80854

Table 5.4: The NDCG scores of Improved ranking algorithm

with enough non-skipped tracks. For the input method 2, it has a more average performance on both the sessions with more skipped tracks and the sessions with more non-skipped tracks, because it learned the features of skipped and non-skipped tracks.

To conclude, the input method 1 has a more strong ability to rank the track with a same type (e.g., the type that is liked by users). In a session, if a user skipped most of the tracks, then there will be no enough tracks to train the features that would be liked by users. However, in this case, there are many skipped tracks, which are thought to be not liked by users. By learning these tracks, the models are expected to output the features that would be disliked by users, i.e., the predicted next track would supposed to be the track that is most likely to be skipped by user, which should be put to the bottom place of a playlist. Thus, following this idea, this project tried to figure out that if the model performance on all groups of test data can be improved by this following ranking algorithm: For sessions with enough non-skipped tracks, the algorithm will get the deep learning model output by giving non-skipped tracks only, and ranking in a descending order according to the output vector score; for sessions without enough non-skipped tracks, the algorithm will get the deep learning model output by giving skipped tracks only, and ranking in a ascending order according to the output vector score (in an order from the tracks with less probability to be disliked to the tracks that are most likely to be disliked). The Combination 1 of input and output methods were applied since it gave the best result on table 5.2. The testing result is shown in table 5.4.

Surprisingly, this NDCG scores was not improved. This indicates that the reverse ranking according to the disliked feature output is not working on the sessions without enough non-skipped tracks. By analysing, this might be because though the non-skipped tracks can be regard as the tracks which are liked by users, the skipped songs are not necessarily disliked by users. There are a lot of reasons for skip, for instance, for a song has been played by a user too many times, even though the user likes all the feature of that song, he might skip it. Some other reasons might be that user can sometimes skip a track by mistake. In addition, even though all the skipped tracks are disliked by user, they can be harder to train than the liked tracks. Because the tracks that are liked by a single user tend to be more uniform style, nevertheless, the styles of tracks disliked by a user can be various, so that the model can be hard to know which features should be learnt. Thus, concluded by this section, the biggest problem with music recommendation at present is to recommend the music for a session with a large numbers of skipped songs.

Chapter 6

Conclusion and Future Works

6.1 project conclusion

To conclude, this project applied the language model to deal with the session-based music sequential tasks, learning users' music preference and automatically ranking the unplayed tracks for them. Overall, the language model is able to works quite efficiently on the sessions with enough non-skipped tracks, however, its effects on the sessions without enough non-skipped tracks have not been proved by this project.

Three neural networks architectures were examined in this project. Based on the result of this project, the Trellis Network performs best on the music sequence tasks, while LSTM has close result to it. RNN is less competent for complex training data.

Compared with the word sequences in NLP, the music sequences need to consider the skip information while training. This project compared two ways to include the skip information. One is to directly delete the skipped tracks from training process, and the other is to concat the session information to track feature vectors. The result shows that the former works better with the sessions with enough non-skipped tracks, while the latter have more average performance on the sessions with and without enough non-skipped tracks.

This project also compared two methods of output. One is to use a decoder to output a vector with a dimension of the length of music library, and each element in that vector corresponds to a single track. The other output a vector with the length of the number of track features. The effects of these two method are similar, with the former performs slightly better.

In addition to answering research question above, this project summarized a rule: while regarding the non-skipped tracks as the tracks that are liked by users, the skipped tracks is not necessary to be disliked by user. The non-skipped tracks are more trainable, and the recommendation based on the skipped tracks should be further researched.

6.2 Further Works

The training time for the models in this project is relatively long, and because there is a limitation on training devices, the model parameter values set in this project were not guaranteed to be the optimal, even after current tuning. For future work, the further tuning is needed to improve the model performance, and the bigger training dataset should be applied to avoid the overfitting problem. In addition, as mentioned in the data analysis part, some of the feature distributions are critical unbalanced, to reduce the influence from bias, data sampling should be considered in the future.

The evaluation metric used in this project is Normalized Discounted Cumulative Gain, however, it seems that it is not susceptible enough and can be hard to move, which causes some problems in comparison of different models. Some other evaluation metrics for ranking algorithm, such as Recall and Precision, Mean Average Precision (MAP) and Expected Reciprocal Rank (ERR) might be consider in the future.

Another problem for this project is that the models seem to be not competent to capture the important elements in each session. As each session was created by different users, the importance of track features can be different in different sessions. For instance, a user may love songs from the 1980s, then the feature 'realise year' might be the most vital feature. If a user want to listen to music for sleep, then the feature 'beat strength' should be the most important elements. The attention mechanism might be useful to realise this function, and should be applied in the future.

As mention in this report, the most difficult part for session-based recommender is the lack of regularity in data, since the data is collected from different users. In addition, the difficulty of user preference prediction can be increased if user provides few non-skipped tracks. Therefore, the future researchers may try to categorize the training data before training. For example, by training the skipped tracks and non-skipped tracks separately, the model performance might be improved. This is just an example, this task deserves more exploration.

Bibliography

[1] J.L. Elman, “Finding Structure in Time,” *Cognitive Science*, vol.14, iss. 2, p.179 - 211, March 1990. [online]. Avaliable: https://www.sciencedirect.com/science/article/pii/036402139090002E. [Accessed August 25, 2019].

[2] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky and S. Khudanpur, “Recurrent neural network based language model,” in *InterSpeech 2010: 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, ISCA Archive, 2010. pp.1045-1048.

[3] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, p.1735-1780, November 1997 . [online]. Avaliable: The MIT Press, https://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735. [Accessed August 25, 2019].

[4] H. Sak, A. Senior, and F. Beaufays, “Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling,” in *INTERSPEECH 2014: 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, ISCA Archive, 2014. pp.338-342.

[5] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” *CoRR*, preprint arXiv:1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555. [Accessed August 25, 2019].

[6] A. Krizhevsky, I. Sutskever and G.E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *NIPS Proceedings: Advances in Neural Information Processing Systems 25, 2012*, F. Pereira, C.J.C. Burges, L. Bottou and K.Q. Weinberge, Eds. USA: Curran Associates, Inc., 2012. pp.1097-1105.

[7] S. Bai, J.Z. Kolter and V. Koltun, “Trellis Networks for Sequence Modeling,” *CoRR*, preprint arXiv:1810.06682v2, 2018. [Online]. Available: https://arxiv.org/abs/1810.06682v2. [Accessed August 25, 2019].

[8] R. Kneser, and H. Ney, “Improved Backing-off for M-gram Language Modelling,” *ICASSP-95: 1995 International Conference on Acoustics, Speech, and Signal Processing, May 9-12, 1995, Detroit, Michigan USA*, IEEE, 1995. pp. 181-184.

[9] Y. Bengio, R. Ducharme, P. Vincent and C. Jauvin, "A Neural Probabilistic Language Model," *Journal of Machine Learning Research 3 (2003)*, p.1137-1155, March 2003. [Online]. Available: http://www.jmlr.org/papers/v3/bengio03a. [Accessed August 25, 2019].

[10] J.S. Bridle, "Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters," in *NIPS Proceedings: Advances in Neural Information Processing Systems 2, 1990*, D.S. Touretzky, Eds. San Francisco: Morgan-Kaufmann, 1990. pp.211-217.

[11] L. Zhu, B. He, M. Ji, C. Ju and Y. Chen, "Automatic Music Playlist Continuation via Neighbor-based Collaborative Filtering and Discriminative Reweighting/Reranking," in *RecSys Challenge: Proceedings of the ACM Recommender Systems Challenge 2018, Vancouver, BC, Canada, October 02 - 02, 2018*, New York: ACM, 2018. Article No. 10.

[12] A. Vandenoord, S. Dieleman and B. Benjamin, "Deep content-based music recommendation," in *NIPS Proceedings: Advances in Neural Information Processing Systems 26, Lake Tahoe, Nevada, December 05 - 10, 2013*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K. Q. Weinberger, Eds. USA: Curran Associates, Inc., 2013. pp.2643-2651.

[13] M. Volkovs, H. Rai, Z. Cheng, G. Wu, Y. Lu and S. Sanner, "Two-stage Model for Automatic Playlist Continuation at Scale,"*RecSys Challenge: Proceedings of the ACM Recommender Systems Challenge 2018, Vancouver, BC, Canada, October 02 - 02, 2018*, New York: ACM, 2018. Article No. 9.

[14] B. Hidasi, A. Karatzoglou, L. Baltrunas and D. Tikk, "Session-based Recommendations with Recurrent Neural Networks," *arXiv*, preprint arXiv: 1511.06939, 2015. [Online]. Available: https://arxiv.org/abs/1511.06939. [Accessed August 25, 2019].

[15] K. Cho, B. van Merrienboer, D. Bahdanau and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv*, arXiv preprint arXiv:1409.1259, 2014. [online]. Available: https://arxiv.org/abs/1409.1259. [Accessed August 25, 2019].

[16] G. Tutz, "Sequential models in categorical regression," *Computational Statistics Data Analysis*, vol.11, no.3, p.275 - 295, May 1991. [online]. Avaliable: Science Direct, https://www.sciencedirect.com/science/article/pii/016794739190086H. [Accessed August 25, 2019].

[17] Y.N. Dauphin, A. Fan, M. Auli and D. Grangier, "Language Modeling with Gated Convolutional Networks," *arXiv*, preprint arXiv:1612.08083, 2016. [Online]. Available: https://arxiv.org/abs/1612.08083. [Accessed August 25, 2019].

[18] S. Bai, J.Z. Kolter and V. Koltun, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," *CoRR*, preprint arXiv:1803.01271, 2018. [Online]. Available: http://arxiv.org/abs/1803.01271. [Accessed August 25, 2019].

[19] S. Merity, N. Sh. Keskar and R. Socher, "An Analysis of Neural Language Modeling at Multiple Scales," *CoRR*, preprint arXiv:1803.08240, 2018. [Online]. Available: https://arxiv.org/abs/1803.08240. [Accessed August 25, 2019].

[20] A. Santoro, R. Faulkner, D. Raposo, J. Rae, M. Chrzanowski, T. Weber, D. Wierstra, O. Vinyals, R. Pascanu, and T. Lillicrap, "Relational recurrent neural networks," in *NIPS Proceedings: Advances in Neural Information Processing Systems 31, 2018*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett, Eds. New York: Curran Associates, Inc., 2018. pp. 7299-7310.

[21] W. Zaremba, I. Sutskever and O. Vinyals, "Recurrent Neural Network Regularization," *CoRR*, preprint arXiv:1409.2329, 2018. [Online]. Available: https://arxiv.org/abs/1409.2329. [Accessed August 25, 2019].

[22] Y. Gal and Z. Ghahramani, "Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference," *CoRR*, preprint arXiv:1506.02158v6, 2016. [Online]. Available: https://arxiv.org/abs/1506.02158v6. [Accessed August 25, 2019].

[23] Y. Gal and Z. Ghahramani, "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks," in *NIPS2016: Advances in Neural Information Processing Systems 29, 2016*, D.D. Lee, M. Sugiyama, U.V. Luxburg, I. Guyon and R. Garnett, Eds. New York: Curran Associates, Inc., 2016. pp. 1019-1027.

[24] S. Merity, N.S. Keskar and R. Socher, "Regularizing and Optimizing LSTM Language Models," *CoRR*, preprint arXiv:1708.02182, 2017. [Online]. Available: http://arxiv.org/abs/1708.02182. [Accessed August 25, 2019].

[25] S. Xie and Z. Tu, "Holistically-Nested Edge Detection,"*ICCV: The IEEE International Conference on Computer Vision, Santiago, Chile, December, 2015*, Computer Vision Foundation, 2015. pp. 1395-1403.

[26] C. Lee, S. Xie, P. Gallagher, Z. Zhang and Z. Tu, "Deeply-Supervised Nets," *CoRR*, preprint arXiv:1409.5185v2, 2014. [Online]. Available: https://arxiv.org/abs/1409.5185v2. [Accessed August 25, 2019].

[27] T. Salimans, D.P. Kingma, "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks," *CoRR*, preprint arXiv:1602.07868, 2016. [Online]. Available: https://arxiv.org/abs/1602.07868. [Accessed August 25, 2019].

[28] Z. Yang, Z. Dai, R. Salakhutdinov and W.W. Cohen, "Breaking the Softmax Bottleneck: A High-Rank RNN Language Model," *CoRR*, preprint arXiv:1711.03953, 2017. [Online]. Available: https://arxiv.org/abs/1711.03953. [Accessed August 25, 2019].

[29] K. Jarvelin and J. Kekalainen, "Cumulated Gain-based Evaluation of IR Techniques," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, p. 422-446, October 2002. [Online]. Available: http://doi.acm.org/10.1145/582415.582418. [Accessed August 25, 2019].

[30] K. Jarvelin and J. Kekalainen, "IR evaluation methods for retrieving highly relevant documents," *SIGIR: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Athens, Greece, 2000*, New York: ACM, 2000. pp. 41-48.

[31] Y. Wang, L. Wang, Y. Li, D. He, W. Chen and T. Liu, *A Theoretical Analysis of Normalized Discounted Cumulative Gain (NDCG) Ranking Measures*, In Proceedings of the 26th Annual Conference on Learning Theory (COLT 2013), 2013. Accessed on: August 25, 2019. [Online]. Available: http://ww.yining-wang.com/colt-ndcg-poster.pdf.

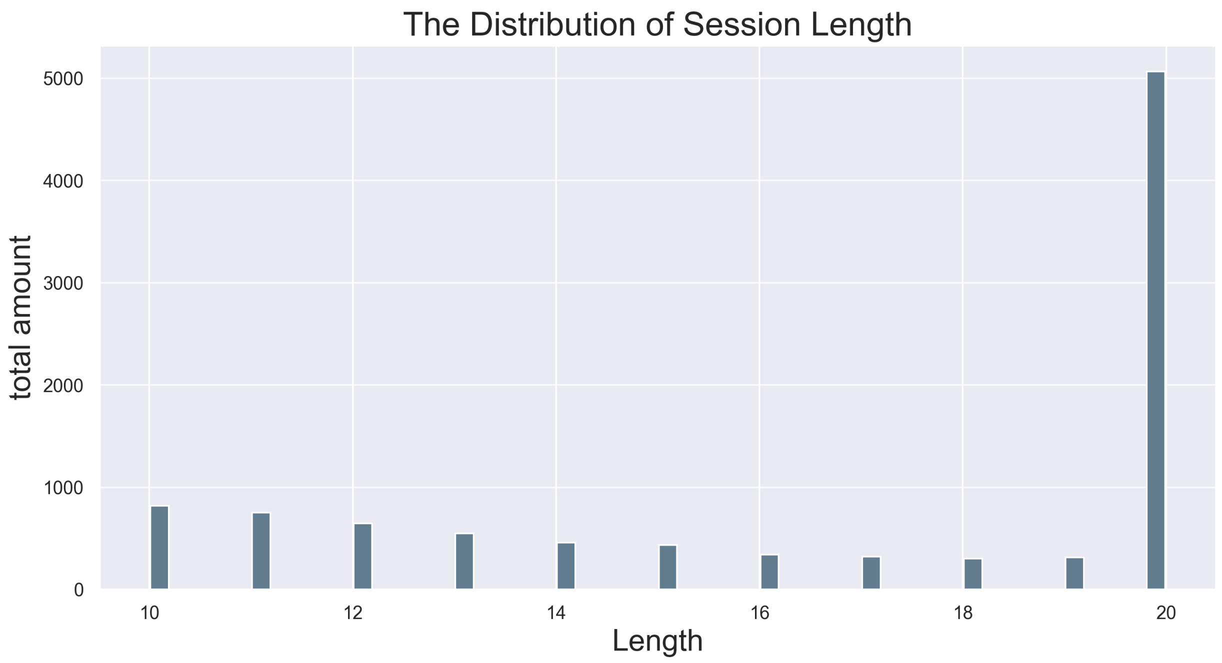
[32] O. Levy and Y. Goldberg, "Dependency-based word embeddings," in *ACL: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, Baltimore, Maryland, USA, 2014*, ACL Anthology, 2014. vol. 2.

[33] L. Bottou, "Large-Scale Machine Learning with Stochastic Gradient Descent," in *COMPSTAT: Proceedings of COMPSTAT'2010, France, August 2010*, Y. Lechevallier and G. Saporta, Eds. Heidelberg: Physica-Verlag HD, 2010. pp.177-186

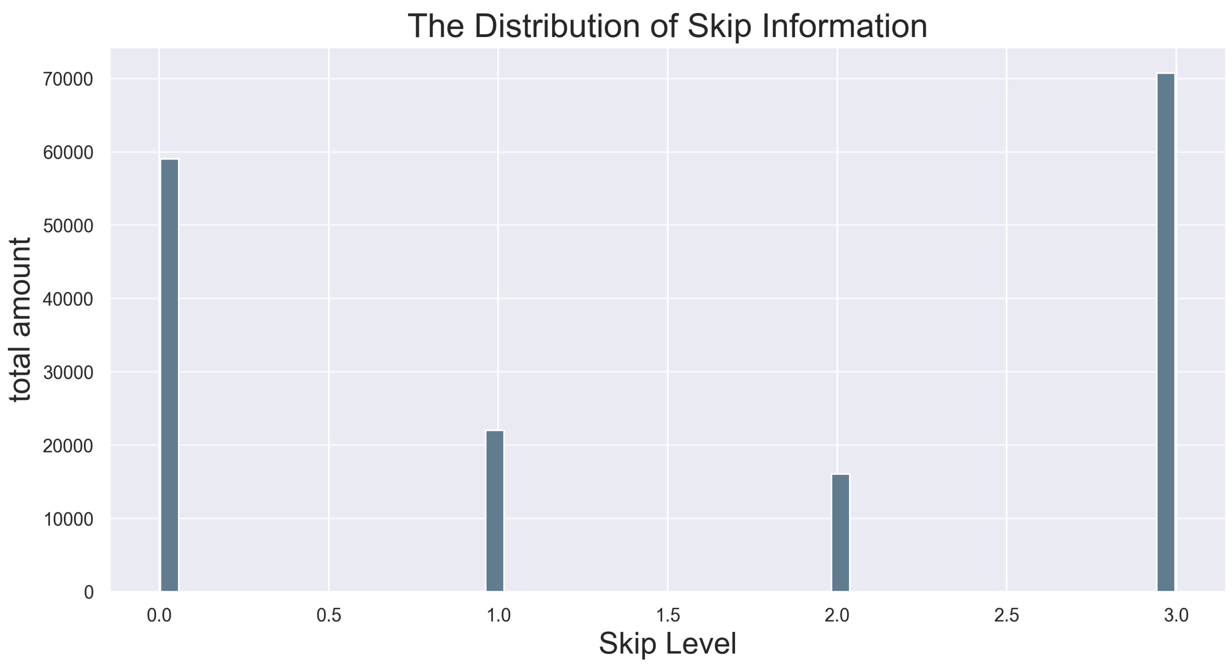
Appendix A

Supplementary Chart and Forms

A.1 Session Length Distribution

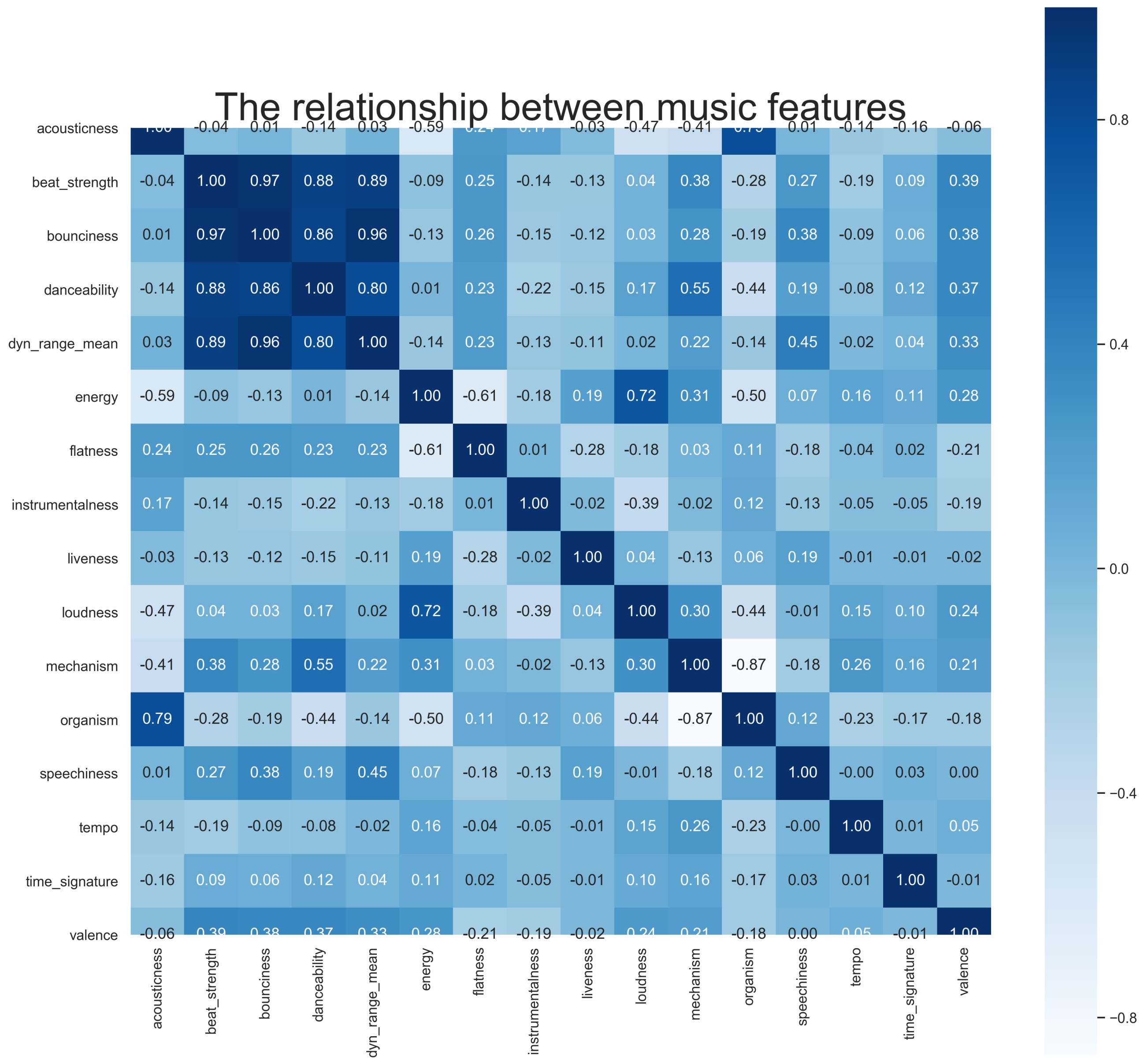


A.2 Session Length Distribution



where ‘0’ means a track was not skipped by users, ‘1’ means a small part of a track was skipped, ‘2’ means a majority of a track was skipped and ‘3’ means a track was skipped at begining.

A.3 Heatmap for correlations between track features



A.4 Standard deviation distribution of release years within a single session

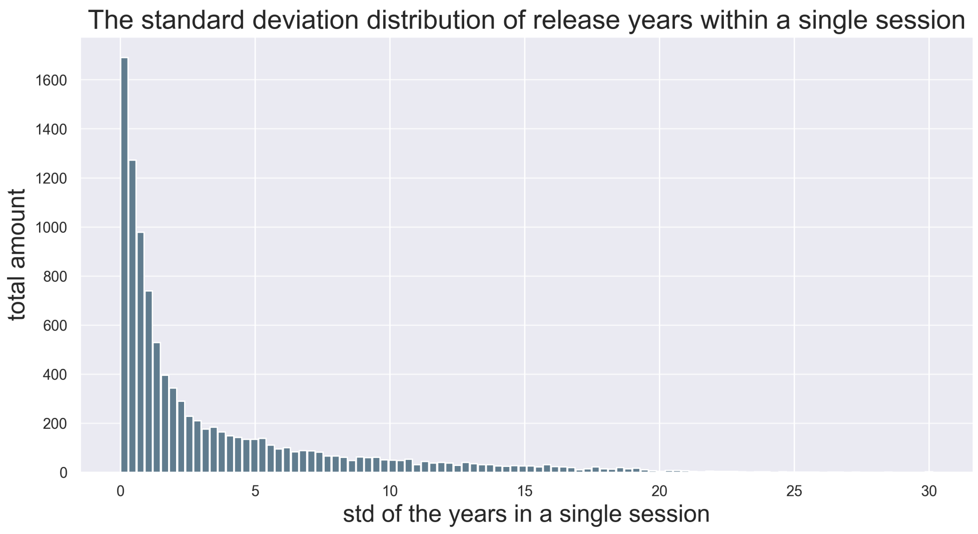


Figure A.1: Std of Release Year for all tracks

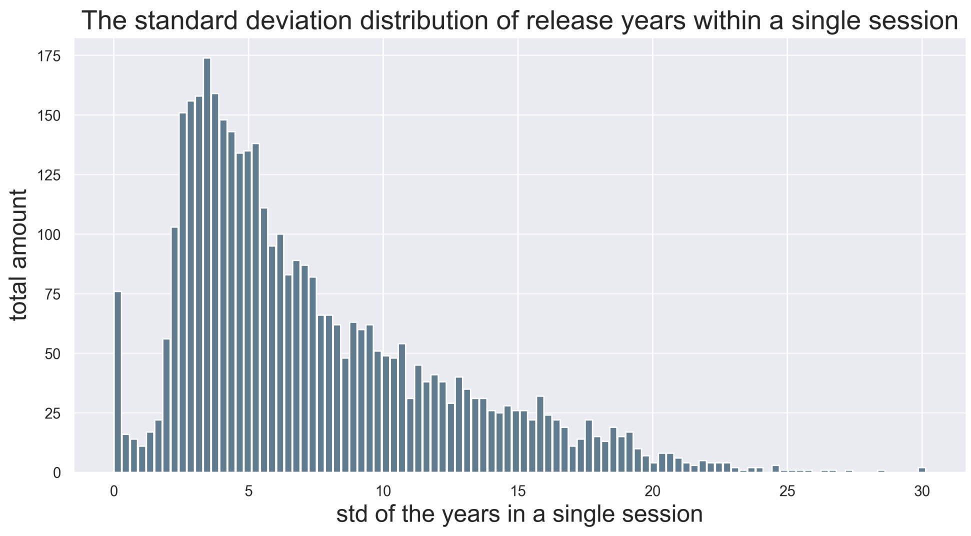


Figure A.2: Std of Release Year for minor tracks (only include sessions with at leat one track that released before 2010))