Personalized Music Explorer: A Hybrid Collaborative Filtering and Deep Learning Recommender-Based Music Generation System

Connor Waldman

June 27, 1018

University of Louisville

J.B. Speed School of Engineering

Department of Computer Engineering and Computer Science

Project Supervisor: Dr. Olfa Nasraoui

Introduction **3**

Executive Summary **3**

System Description **4**

Needs Assessment 4

System Specification (Software Design Document) 4

System Diagrams 8

Hardware/Software Overview 8

Constraints 10

Detailed Implementation **10**

Hardware and Software Requirements Specifications 10

Test/Evaluation Experimental Procedure **14**

Societal Impact **23**

Legal and Ethical Considerations 23

Contribution and Effects **24**

Conclusions **24**

References 25

Introduction

Music is intrinsically shaped by human interaction. Tradition, culture, and environment are all factors which have influenced how music is composed and listened to. With any creative form of expression, such as visual art or creative writing, influence is omnipresent. Machine learning has been implemented in almost every facet of art, and although it may never replace human composition, it can aid the process and understanding of creativity.

This report describes a multi-model song recommender and music generation system which creates original compositions based off a user's listening behavior. The song recommender system uses *Collaborative Filtering* to recommend a specific amount of songs to a user. The music generation system uses *Content Based* machine learning to create original pieces of music, influenced by MIDI music content data. Both categories of recommender systems have benefits and trade-offs; using these two systems together creates a hybrid recommendation system which capitalizes from the benefits each model provides. Using these models together can help users from multiple disciplines understand the nature of artistic expression, and how further comprehension of influence in music leads to new discoveries.

Executive Summary

This report will describe the components and functions of the Personalized Music Explorer. It describes in detail the three modules used in this project, they are: *Recommender System, MSD2MIDI,* and *Neural Composer.* The *Recommender System* uses the Million Song Dataset to obtain user, song, and listening data for personalized music recommendations. Specifically it uses Singular Value Decomposition as the method of generating song recommendations. The *MSD2MIDI* module confidently matches the recommended songs from the *Recommender System* to the corresponding MIDI files. These MIDI files are then used in the *Neural Composer*, which uses Long Short-Term Memory (LTSM), a type of Recurrent Neural Network (RNN), to generate new music via MIDI, influenced by the input MIDI data.

The experimental process used to evaluate the project, and its potential, focused on using six different sets of MIDI data to train the network. For comparative purposes, two of the sets were acquired through the recommender system, and four were manually assembled. The four manually assembled MIDI sets sought to generate different output and obtain knowledge from patterns present in the music.

Recommender systems have become a major part of almost every modern-day industry that interacts with users online. In online sales, they match users with items, alleviating the overwhelming amount of products available. These systems estimate relevance between a user and item based off available data, creating *personalized* recommendations. The Personalized Music Explorer utilizes two methods in recommender systems to generate music based off user listening history.

Collaborative Filtering allows the recommender system to discover completely new items the user wouldn't normally discover through inductively learning patterns through user data. In this project, the *Recommender System* uses Collaborative Filtering to recommend songs the user would not normally listen to, based off song play counts and other user listening history. A downside to this method is that it cannot cold start, meaning new users and songs have no history associated with them, therefore cannot be recommended to users.

Content Based Filtering solves the cold start shortcoming of Collaborative Filtering by determining and classifying neighborhoods of content. This is especially useful for using sequential MIDI data, which has no other associated data, but contains information (In this case, sequences of notes) which can be classified and replicated.

Using these two methods of classification creates a useful hybrid model which successfully implements the modules of the Personalized Music Explorer. Through the experiments conducted in this project, the system's potential applications are revealed, and lays a foundation for other users to learn and expand the Personalized Music Explorer.

System Description

Needs Assessment

Composition is the art of making decisions, including the decision not to do so. Currently, a range of tools exist in which music can be created with minimal interaction and input, but few consider the user's personal taste. Modern-day sound design technology grants the ability to create any imaginable sound, which has added significant weight to the task of writing music. Composers who predated computers had much fewer parameters to consider, and yet still they sought methods of relieving the responsibility of making decisions. Before computers were widely available to the public, composers found ways to generate music, both stochastically and autonomously. Greek composer Iannis Xenakis created music based on Markov chains in his piece *Analogiques*, John Cage developed aleatoric methods to decide parameters of a piece, both with the goal of removing oneself from the compositional process. Utilizing recommendation systems and machine learning can take these concepts a step further. Rather than using methods which make uneducated decisions toward the outcome of a piece of music, a personalized music generator has the ability to determine parameters with the user's preference in mind, while still removing them from the process of composition.

Systems of this nature are not exclusive to composers, though. Combining recommendation systems with music generation allows for discovery of new patterns in music. Researchers can use it as a tool to further understand listener's preferences in music, and reveal trends in society in regard to musical culture. The Personalized Music Explorer provides a basis for these discoveries to be made.

System Specification (Software Design Document)

1.1 Purpose

This section describes the design for the Personal Music Explorer system.

1.2 Scope

The design covered in this section includes the recommender system, Million Song Dataset-to-MIDI module, and the music generation system.

1.3 Definitions, Acronyms, and Abbreviations

Description	**Abbreviation**
Million Song Dataset	MSD
Recurrent neural network	RNN
Long short-term memory	LSTM

2. References

Million Song Dataset: https://labrosa.ee.columbia.edu/millionsong/

Lakh MIDI Dataset: http://colinraffel.com/projects/lmd/#learn

3. Module Design

Personalized Music Explorer is best represented through the modules it consists of. It consists of three essential modules. Respectively, they are the *Recommender System*, *MSD2MIDI*, and *Neural Composer*.

3.1 Module Detailed Design for Personalized Music Explorer

3.1.1 *Recommender System*

This component will be the first module required to run Personalized Music Explorer. It loads the required data and metadata for viewing and generating recommendations, and creates data subsets appropriate for the model. It provides the recommendation engine feature and displays a specified number of recommended songs per user id.

3.1.2 *MSD2MIDI*

This module matches the MSD ids with the Lakh MIDI Dataset ids, giving the user access to MIDI files which correspond to each recommended song. It also creates visualizations of the midi data, plotting MIDI notes for entire songs or for a specific instrument.

3.1.3 *Deep Neural Composer*

Using a Recurrent Neural Network with Long-Short-Term Memory (LSTM)[3], this module parses the MIDI files, prepares the data for the model, and generates a new music, via a MIDI file.

3.2 Data Properties

3.2.1 The Million Song Dataset is a collection of audio features and metadata for one million contemporary popular music tracks. Used specifically, is the Taste Profile Subset, a dataset which provides real user play counts from undisclosed partners of MSD. The dataset provides triplet data consisting of user id, song id, and play count. The available metadata is also used to acquire additional information, including: song title, song duration, artist name, and year released.

3.2.2 The Lakh MIDI Dataset is a collection of 176,581 MIDI files. 45,129 files have been matched to entries in the MSD. It also provides a JSON file which contains match confidence scores for each matched pair. This project uses the 45,129 files in the matched data subset.

3.3 State Model Decomposition

Personalized Music Explorer consists of the states shown in Figure 1.1

3.4 Use Case Model Decomposition

Personalized Music Explorer consists of the use cases for a general user and a developer (Shown in Figure 1.2)

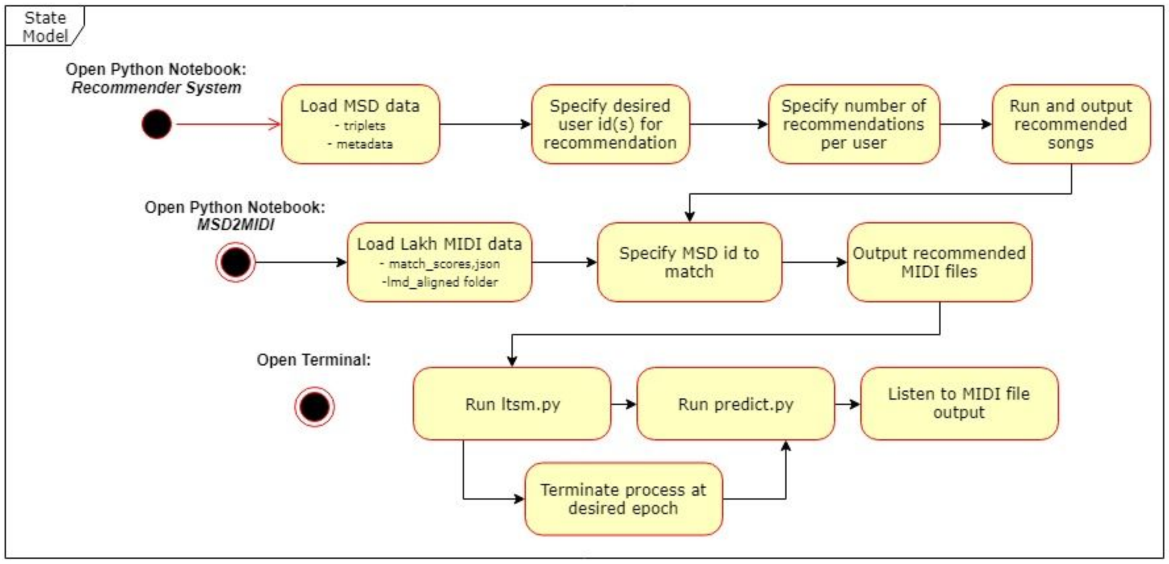

State Model
Open Python Notebook: *Recommender System*
Load MSD data - triplets - metadata
Specify desired user id(s) for recommendation
Specify number of recommendations per user
Run and output recommended songs
Open Python Notebook: *MSD2MIDI*
Load Lakh MIDI data - match_scores.json -lmd_aligned folder
Specify MSD id to match
Output recommended MIDI files
Open Terminal:
Run ltsm.py
Run predict.py
Listen to MIDI file output
Terminate process at desired epoch

Figure 1.1 – State Model for Personalized Music Explorer

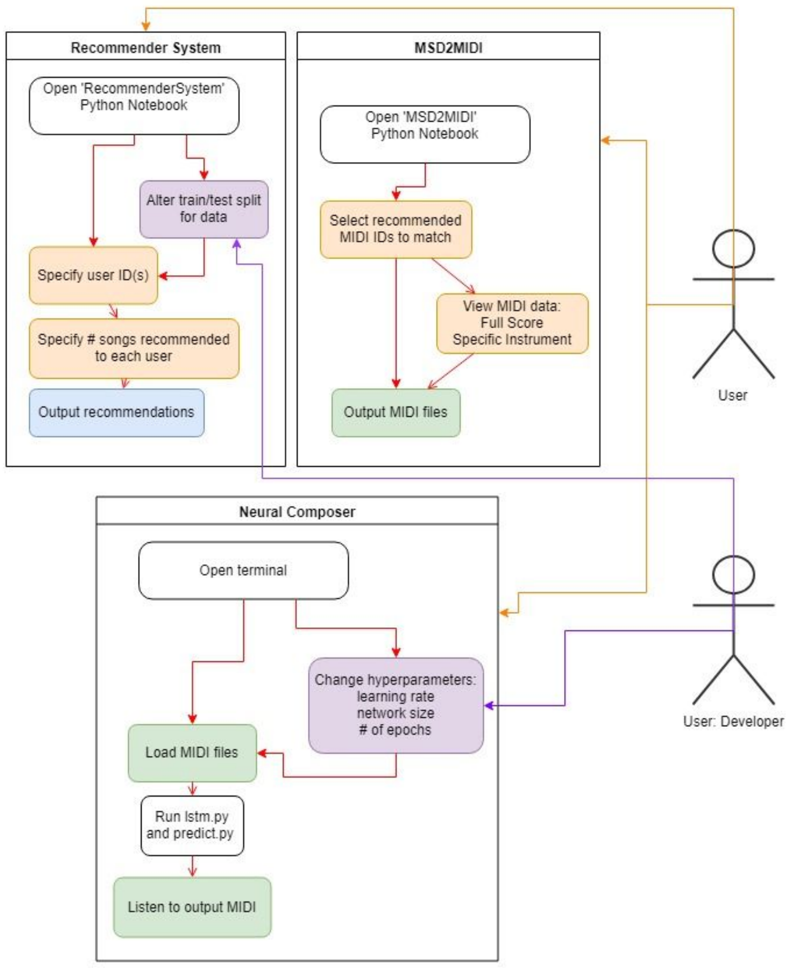

Recommender System
Open 'RecommenderSystem' Python Notebook
Alter train/test split for data
Specify user ID(s)
Specify # songs recommended to each user
Output recommendations
MSD2MIDI
Open 'MSD2MIDI' Python Notebook
Select recommended MIDI IDs to match
View MIDI data: Full Score Specific Instrument
Output MIDI files
User
Neural Composer
Open terminal
Change hyperparameters: learning rate network size # of epochs
Load MIDI files
Run lstm.py and predict.py
Listen to output MIDI
User: Developer

Figure 1.2 - Use Case Model

4. Interface description

4.1 Module interfaces

4.1.1 Interface to the *Recommender System*

The interface is presented via Google Colaboratory, a Jupyter notebook environment which runs in Google's cloud system. The *Recommender System* utilizes the Python3 and GPU runtime available in Colaboratory, and presents each section of code with descriptive text, along with instructions on how to use it. Every cell of code can be run or rerun at any time.

4.1.1.1 Recommend – Last cell in the notebook, outputs specified amount of recommendations for specified user(s) in a text list.

4.1.2 Interface to *MSD2MIDI*

Rather than on a Google Colaboratory notebook, *MSD2MIDI* is accessed through a local Jupyter notebook environment, in order to access the large dataset of MIDI files stored on a local drive. There are three interfaces, presented as code cells, in this environment:

4.1.2.1 Match Dataset – Using the JSON file “match_scores”, the Lakh MIDI Dataset contents are matched and aligned to entries in the MSD. Upon successful matching, the MIDI file id is displayed with the specified MSD ID (msd_id).

4.1.2.2 Transcribe Score – Displays a visualization of the MIDI piano roll score as Note(s) vs. Time.

4.1.2.3 Transcribe Instrument – Displays a visualization of a specified MIDI instrument as Note(s) vs. Time.

4.1.3 Interface to *Neural Composer*

Neural Composer is run via terminal on a local machine. The interface consists of running two commands, which can executed after navigating to the appropriate directory:

4.1.3.1 lstm.py – After loading desired midi files into the folder “midi_songs”, the execution of this command trains the data. It displays the current number of epoch running, along with the time left before completion of the epoch, and current loss. It also displays past epochs run, and their respective number, time run, and loss.

4.1.3.2 predict.py – After specifying the correct “weight.hdf5” file to use. This command generates and outputs a MIDI file titled “test_output” of the generated music.

System Diagrams

See External Design Document

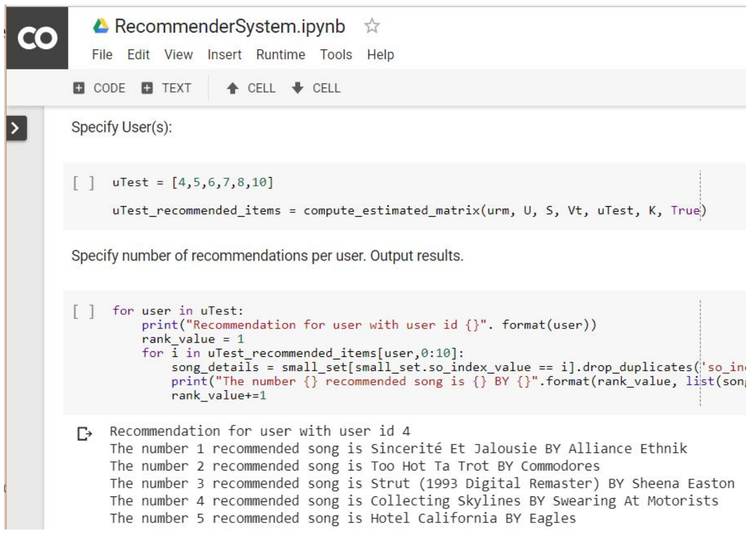
Hardware/Software Overview

1. **Hardware**

The hardware used was a laptop computer running Windows 10, with 8 GB RAM and Intel i7 process at 2.70 GHz.

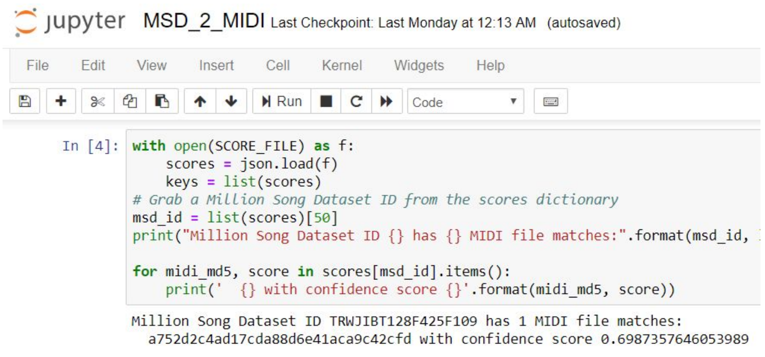
2. **Software**

For the *Recommender System*, Google Chrome was running Google Colaboratory with a Python3 runtime and GPU hardware accelerator (Figure 1.1). The *MSD2MIDI* module was run via the Anaconda Prompt interface, with a Jupyter notebook in Google Chrome (Figure 1.2). The *Neural Composer* module was run by an Anaconda Prompt (Figure 1.3).



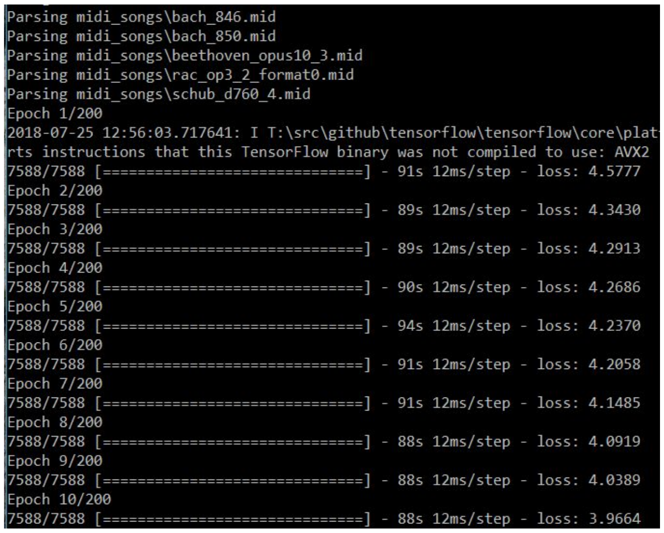
```
Specify User(s):

[ ]  uTest = [4,5,6,7,8,10]

     uTest_recommended_items = compute_estimated_matrix(urm, U, S, Vt, uTest, K, True)

Specify number of recommendations per user. Output results.

[ ]  for user in uTest:
         print("Recommendation for user with user id {}". format(user))
         rank_value = 1
         for i in uTest_recommended_items[user,0:10]:
             song_details = small_set[small_set.so_index_value == i].drop_duplicates('so_in
             print("The number {} recommended song is {} BY {}".format(rank_value, list(son
             rank_value+=1

     Recommendation for user with user id 4
     The number 1 recommended song is Sincerité Et Jalousie BY Alliance Ethnik
     The number 2 recommended song is Too Hot Ta Trot BY Commodores
     The number 3 recommended song is Strut (1993 Digital Remaster) BY Sheena Easton
     The number 4 recommended song is Collecting Skylines BY Swearing At Motorists
     The number 5 recommended song is Hotel California BY Eagles
```

Figure 1.1 - Recommender System: Google Colab Notebook



```
In [4]: with open(SCORE_FILE) as f:
            scores = json.load(f)
            keys = list(scores)
        # Grab a Million Song Dataset ID from the scores dictionary
        msd_id = list(scores)[50]
        print("Million Song Dataset ID {} has {} MIDI file matches:".format(msd_id,

        for midi_md5, score in scores[msd_id].items():
            print('  {} with confidence score {}'.format(midi_md5, score))

        Million Song Dataset ID TRWJIBT128F425F109 has 1 MIDI file matches:
          a752d2c4ad17cda88d6e41aca9c42cfd with confidence score 0.6987357646053989
```

Figure 1.2 - MSD2MIDI: Jupyter notebook



```
Parsing midi_songs\bach_846.mid
Parsing midi_songs\bach_850.mid
Parsing midi_songs\beethoven_opus10_3.mid
Parsing midi_songs\rac_op3_2_format0.mid
Parsing midi_songs\schub_d760_4.mid
Epoch 1/200
2018-07-25 12:56:03.717641: I T:\src\github\tensorflow\tensorflow\core\plat
rts instructions that this TensorFlow binary was not compiled to use: AVX2
7588/7588 [==============================] - 91s 12ms/step - loss: 4.5777
Epoch 2/200
7588/7588 [==============================] - 89s 12ms/step - loss: 4.3430
Epoch 3/200
7588/7588 [==============================] - 89s 12ms/step - loss: 4.2913
Epoch 4/200
7588/7588 [==============================] - 90s 12ms/step - loss: 4.2686
Epoch 5/200
7588/7588 [==============================] - 94s 12ms/step - loss: 4.2370
Epoch 6/200
7588/7588 [==============================] - 91s 12ms/step - loss: 4.2058
Epoch 7/200
7588/7588 [==============================] - 91s 12ms/step - loss: 4.1485
Epoch 8/200
7588/7588 [==============================] - 88s 12ms/step - loss: 4.0919
Epoch 9/200
7588/7588 [==============================] - 88s 12ms/step - loss: 4.0389
Epoch 10/200
7588/7588 [==============================] - 88s 12ms/step - loss: 3.9664
```

Figure 1.3 - Neural Composer on Anaconda Prompt

Constraints

This project consisted of economical, technical, and time constraints. All frameworks, software, data, and services used were used at no additional cost, with the intention that users of the Personalized Music Explorer would not have to pay an additional fee to use the system. Although Google Colaboratory allows for GPU hardware accelerators to be used, this limited the project to using CPU for the *MSD2MIDI* module, which is run on a local Jupyter notebook.

The experiments conducted in this project accounted for time and technical limitations. Generally speaking, more training epochs will result in better fitting of the data. Running a large number of epochs, which produced weight files approximately 41 MB per epoch, was taxing on the laptop running these tests. There is also a limitation in results due to not having a systematic evaluation based on a large varied test. This largely in part due to the difficulty of evaluating music.

The length of output MIDI files was originally around 2 minutes per file. This was determined to be unnecessarily long, and taking into account a 2017 study by Hubert Léveillé Gauvin, the average attention span for a piece of music presently is 5 seconds long compared to 20 seconds in 1980. Averaged out, this came to 15 seconds, which is ideal in ensuring users listen to the entire generated song. This limited the output a file to have a sequence length of 50 notes, which output approximately 15 seconds of music.

Detailed Implementation

Hardware and Software Requirements Specifications

1.1 Purpose
This document outlines all of the hardware and software requirements for the Personalized Music Generator. Parts 1 and 2 are intended for users and customers of the application, but are also applicable to software engineers and designers. Part 3 is mainly intended for software engineers, but can also be of use to customers and users.

1.2 Scope
This document includes the requirements for the three systems required to run Personalized Music Explorer. This release is intended for PC, laptop, and desktop computers.

2. Overview
The remainder of this section includes two parts. The first of the two is an overview of the product perspective. The second specifies functional, performance, and database requirements and attributes. This section gives an overview of the whole system, and explains how each of the separate systems interact with each other.

2.1 Product perspective

The current state of Personalized Music Explorer is for composers/musicians, teachers, developers, and researchers. It is mainly intended for adults who have a general knowledge of how to work with python notebooks and run terminal commands. It is also designed for users and developers to make adjustments to model hyperparameters.

2.1.1 System interfaces

Microsoft Windows 7 or later, macOS 10.12.6 (Sierra) or later, or Ubuntu 16.04 or later. Minimum of an Intel i3 processor

2.1.2 User interfaces

Monitor, Keyboard, and Mouse

2.1.3 Hardware interfaces

PC, laptop, and desktop computers

2.1.4 Software interfaces

Internet browser (Google Chrome recommended); Terminal (Anaconda Prompt recommended); Python3

2.1.5 Communications interfaces

Personalized Music Explorer requires a stable Internet connection.

2.1.6 Memory Constraints

Machines running Personalized Music Explorer should have at least 10 GB of free storage and a minimum of 4 GB RAM.

2.2 System functions

With the Personalized Music Explorer, the user will be able to generate song recommendations from a specified user ID. They will also have the ability to match the recommended song IDs to a corresponding MIDI file. The *Neural Composer* will be able to use any MIDI data, including the matched recommended data, and train the network to produce a new generated MIDI file. The user will be able to locate this file and play it.

2.3 User characteristics

The user is expected to be knowledgeable and mature enough to operate computers.

2.4 Constraints, assumptions, and dependencies

Personalized Music Explorer will run on a computer. Python will be the implementation language.

3. Specific requirements

3.1 External Interfaces

3.1.1 User Interfaces:

3.1.1.1 Internet browser: a user should have access to an up-to-date internet browser with the capability to run python notebooks:

3.1.1.1.1 Google Colaboratory Notebook

3.1.1.1.2 Jupyter Notebook

3.1.1.2 Terminal

3.2 Functional Requirements

3.2.1 Recommender System

3.2.1.1 Import file: necessary to load datasets, metadata, python classes and packages.

3.2.1.2 Create subset: perform operations to create compatible dataset dimensions for matrix factorization

3.2.1.3 Add metadata: allows user to display additional information about the database. Most importantly the "song_id" which is necessary for the MSD2MIDI module.

3.2.1.4 View dataset: allows user to display current dataset, including original dataset, generated subsets, and metadata.

3.2.1.5 Perform SVD: allows user to apply Singular Value Decomposition to the dataset for recommendation generation.

3.2.1.5 Specify user: allows user to alter the specific user ID and amount of users to generate recommendations

3.2.1.6 Specify recommendations: allows user to specify the number of song recommendations per user.

3.2.1.7 Display recommendation(s): displays recommended song titles, associated artists, and song IDs for each specified user.

3.2.2 MSD2MIDI

3.2.2.1 Import data: necessary for loading "match_scores.json" and the aligned MIDI data subset. Must use "matched" subset, which has verified correct MIDI file matches.

3.2.2.2 Specify song ID: specify MSD ID to be matched to corresponding MIDI file.

3.2.2.3 Match Dataset: perform operation which matches the MSD ID to the correct MIDI file.

3.2.2.4 Display MIDI score: allows user to visualize data from plotting MIDI data by Notes vs Time, specifically the entire score of the piece.

3.2.2.5 Display MIDI instrument: allows user to visualize data from plotting MIDI data by Notes vs Time, specifying a particular instrument to be displayed.

3.2.3 Neural Composer

3.2.3.1 Load MIDI files: allows user to load any amount of MIDI files to the system to be converted to an array and parsed

3.2.3.2 Split data: splits data into two object types: notes and chords, with notes containing pitch (frequency of sound), octave, and offset (where the note is located in the piece)

3.2.3.3 Map function: map string-based categorical data to integer-based numerical data. Allows for improved performance of neural network model, because it handles numerical data better than string-based data.

3.2.3.4 Modify learning rate: allows user to specify learning rate (set as "sequence_length") to alter the amount of notes the network requires before making a prediction.

3.2.3.5 Network layers: allows user to specify type and number of layers in the network. The default layers will be LSTM, Dropout, Dense, and Activation layers.

3.2.3.6 Specify number of epochs: allows user to alter the amount of epochs run for training

3.2.3.7 Model checkpoints: save the weights of the network with each epoch completion, even while model is being run. Allows user to stop training at any point in the training and output generated MIDI.

3.2.3.8 Visual feedback: Display loss function, time and number of epochs run while training network.

3.2.3.9 Output MIDI file: output a midi file titled "test_output" after predict.py has completed running.

3.3 Performance Requirements

3.3.1 Available data: the system shall provide available and compatible data for use with the modules.

3.3.2 Comprehensive visual feedback: the system shall provide valuable feedback to the user, both in visual or graphical representations and text.

3.3.3 Proper formats: the generated MIDI files shall be output in correct format, so a user can play them promptly

3.4 Design Constraints

3.4.1 Internet reliability: the system requires an internet connection, internet performance and speeds are not guaranteed.

3.4.2 External services: Google applications and Anaconda cloud service performance are out of the systems scope of control.

3.5 Software System Attributes

3.5.1 Reliability
Personalized Music Explorer shall not fail to run any modules of the system more than 1/1000 of the times run.

3.5.2 Availability
Out of 1000 hours of use, Personalized Music Explorer shall be available more than 98% of the time

3.5.3 Security
User must have active Google account and log in under a secure network. User data used in MSD is verified private by The Echo Nest.

3.5.4 Maintainability
The system is designed to be frequently altered. A main purpose of the architecture is to allow others to build off the existing functionality.

3.5.4. Portability
The application shall be portable with any laptop. Every component of the system is available online, giving it the ability to be run on any computer which meets the requirements.

Test/Evaluation Experimental Procedure

The main purpose of this experimental procedure was to use varying styles and combinations of MIDI songs to derive further knowledge about the relationship between recommender systems and music generation. It also sought to learn from the results and explore different possible uses for this system that may benefit certain parties, such as composers, teachers, students, developers, and others which may not have been originally accounted for.

1. Evaluation - Objective Methods

1.1 The Dataset:

1.1.1 *Recommender System*

The matched MSD to MIDI dataset is necessary for the purpose of the system. Only songs which have been confidently matched to a corresponding MIDI file can be used for the *Neural Composer*, which takes MIDI files as input to train the recurrent neural network (RNN). Because of this requirement, the rest of the dataset was left out.

- Million Song Dataset: ~1,000,000 songs
- Recommendations using matrix factorization only require user, item, and rating. The Taste Profile subset in the form of triplet data fulfills this requirement. It consists of <u>384,546 unique songs</u>.
- Lakh MIDI Dataset: 176,581 unique MIDI files
    - <u>45,129 confidently matched to MSD IDs</u>
- Train/Test Split (see Figure 1.1):
    - Train: 90% of the data
    - Test: 10% of the data

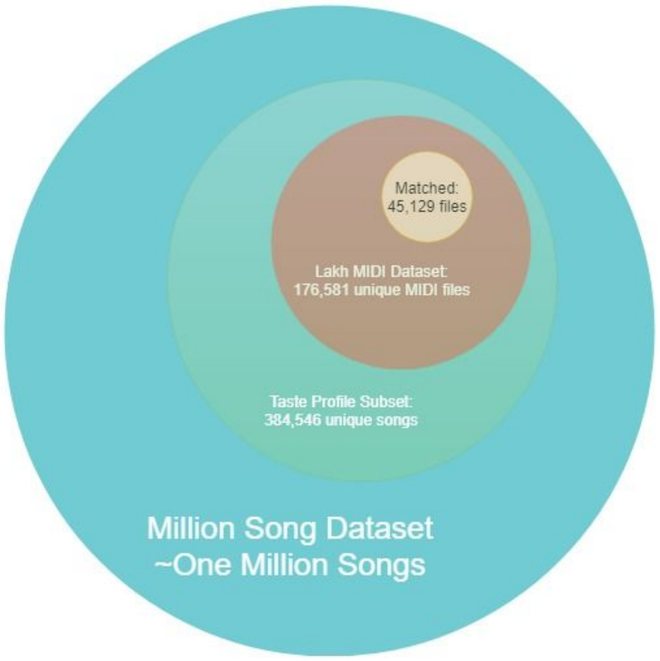


Figure 1.1

1.1.2 Matrix Factorization: Singular-Value Decomposition

Due to the availability of a large amount of user data, the goal is to recommend songs based off user ratings. To do this, we capture the latent relationships between users and songs, which allow us to compute the predicted likeliness of a certain users. Then, we use SVD to produce a two-dimensional representation of the original user-song matrix, and then compute the neighborhood in the reduced space. By reducing the dimensionality of the space, we can increase density to find more ratings and discover latent relationships.

SVD is a matrix factorization technique. With *m* users and *n* songs, SVD factors an *m* x *n* matrix **X** into three matrices: **U**, **S**, and $\mathbf{V}^T$ shown in Figure 1.2. Matrices **U** and **V** are of size *m* x *r* and *n* x *r*, respectively; *r* is the rank of the matrix **X**. The rank is the number of linearly independent rows or columns in the matrix **X. S** is a diagonal matrix of size *r* x *r* having all singular values of matrix **X** as its diagonal entries. All the entries of matrix S are positive and stored in decreasing order of their magnitude.

$$\overset{X}{\underset{m\times n}{\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n}\\ x_{21} & x_{22} & \cdots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn}\end{pmatrix}}} = \overset{U}{\underset{m\times r}{\begin{pmatrix} u_{11} & \cdots & u_{1r}\\ \vdots & \ddots & \\ u_{m1} & & u_{mr}\end{pmatrix}}} \overset{S}{\underset{r\times r}{\begin{pmatrix} s_{11} & 0 & \cdots\\ 0 & \ddots & \\ \vdots & & s_{rr}\end{pmatrix}}} \overset{V^{\mathsf{T}}}{\underset{r\times n}{\begin{pmatrix} v_{11} & \cdots & v_{1n}\\ \vdots & \ddots & \\ v_{r1} & & v_{rn}\end{pmatrix}}}$$

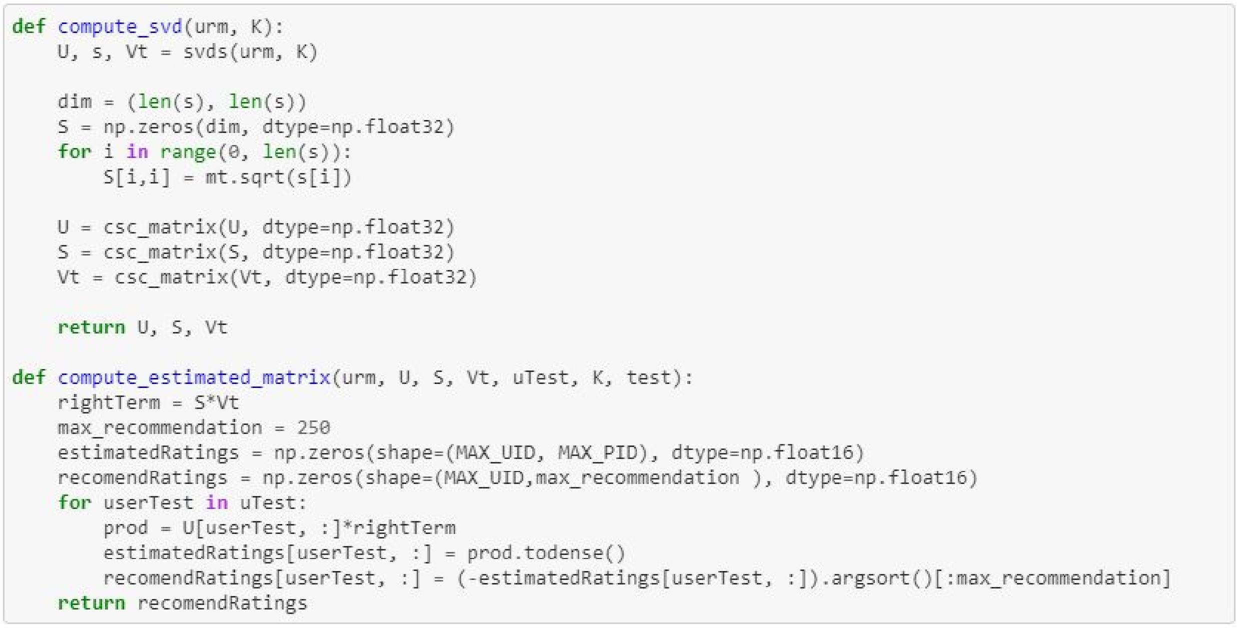
Figure 1.2

We then want to compare songs based off user play count. To accomplish this, with *k* being the number of top ordered recommendations we keep the first *k* singular values in **S**, where *k* < *r*, giving us the best rank *k* approximation to **X**, reducing the dimensionality of the original space. This is shown in Figure 1.3:

$$\overset{\hat{X}}{\underset{m\times n}{\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n}\\ x_{21} & x_{22} & \cdots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn}\end{pmatrix}}} \approx \overset{U}{\underset{m\times r}{\begin{pmatrix} u_{11} & \cdots & u_{1r}\\ \vdots & \ddots & \\ u_{m1} & & u_{mr}\end{pmatrix}}} \overset{S}{\underset{r\times r}{\begin{pmatrix} s_{11} & 0 & \cdots\\ 0 & \ddots & \\ \vdots & & s_{rr}\end{pmatrix}}} \overset{V^{\mathsf{T}}}{\underset{r\times n}{\begin{pmatrix} v_{11} & \cdots & v_{1n}\\ \vdots & \ddots & \\ v_{r1} & & v_{rn}\end{pmatrix}}}$$

Figure 1.3

In the *Recommender System*, it is implemented in python as the following function (Figure 1.4):



```python
def compute_svd(urm, K):
    U, s, Vt = svds(urm, K)

    dim = (len(s), len(s))
    S = np.zeros(dim, dtype=np.float32)
    for i in range(0, len(s)):
        S[i,i] = mt.sqrt(s[i])

    U = csc_matrix(U, dtype=np.float32)
    S = csc_matrix(S, dtype=np.float32)
    Vt = csc_matrix(Vt, dtype=np.float32)

    return U, S, Vt

def compute_estimated_matrix(urm, U, S, Vt, uTest, K, test):
    rightTerm = S*Vt
    max_recommendation = 250
    estimatedRatings = np.zeros(shape=(MAX_UID, MAX_PID), dtype=np.float16)
    recomendRatings = np.zeros(shape=(MAX_UID,max_recommendation ), dtype=np.float16)
    for userTest in uTest:
        prod = U[userTest, :]*rightTerm
        estimatedRatings[userTest, :] = prod.todense()
        recomendRatings[userTest, :] = (-estimatedRatings[userTest, :]).argsort()[:max_recommendation]
    return recomendRatings
```

Figure 1.4 - SVD function in Python

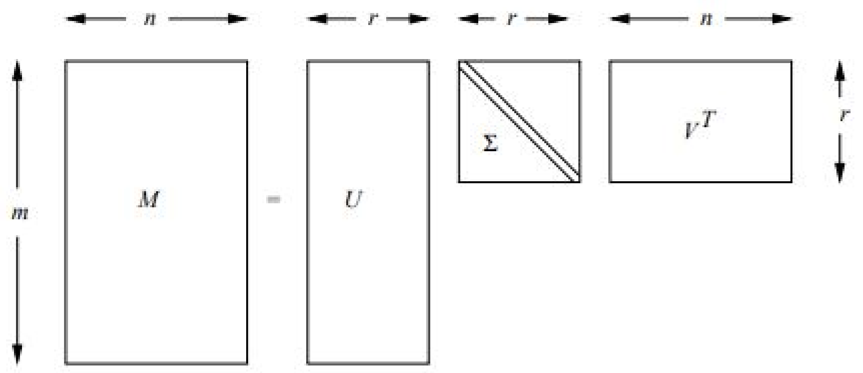


Figure 1.5 - The form of SVD

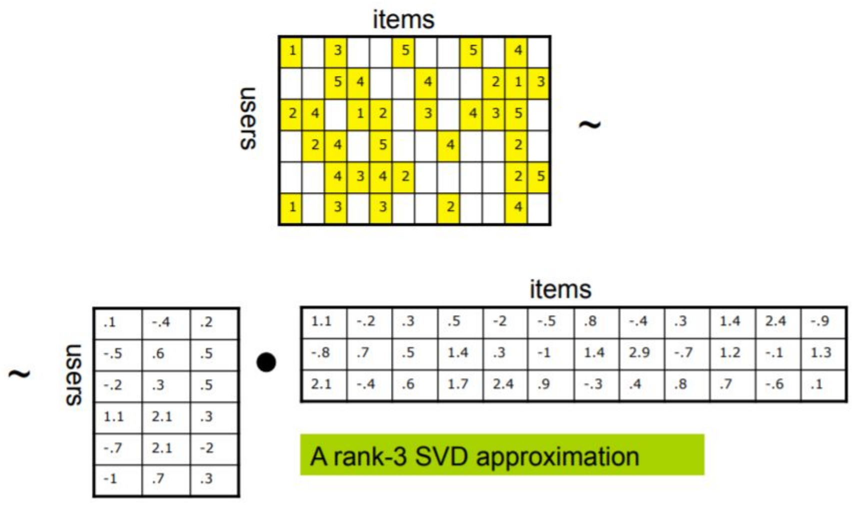


Figure 1.6 - Basic matrix factorization model.

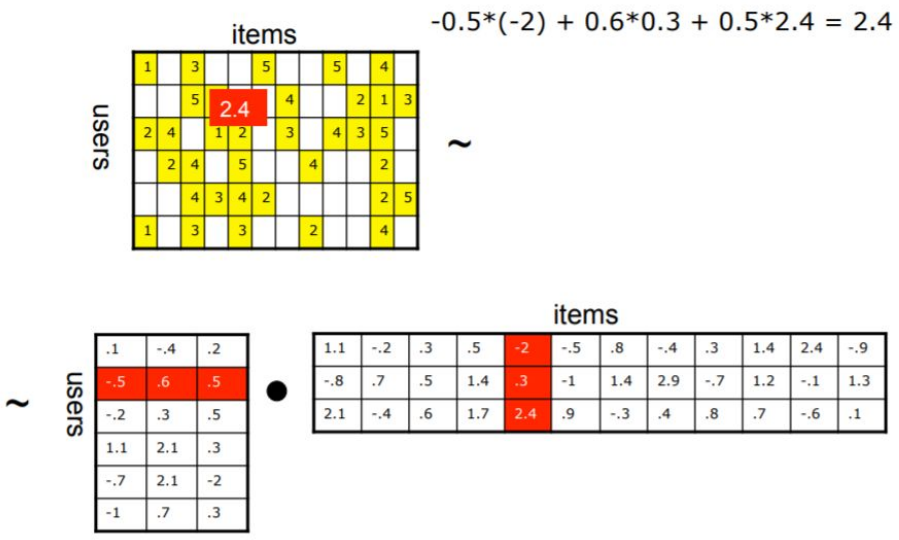
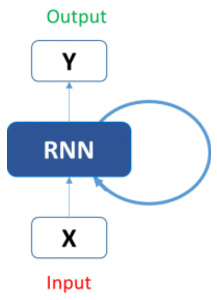


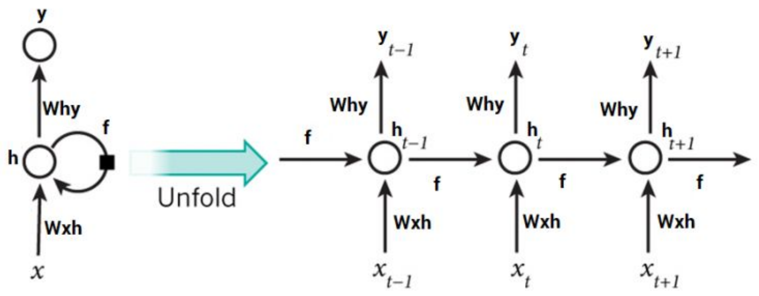
Figure 1.7 - Estimating unknown ratings.

`1.1.2 *Neural Composer*

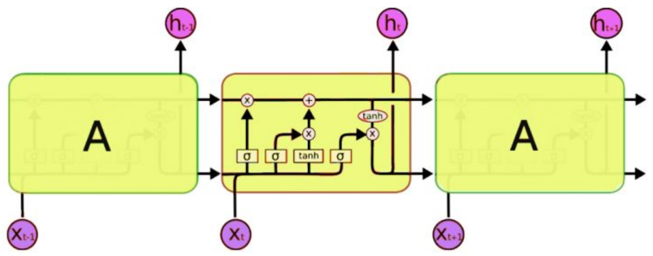
Due to the sequential nature of MIDI data, the *Neural Composer* depends largely on the Long Short-Term Memory (LSTM) model. LSTMs are able to recognize and encode long-code patterns using a gating mechanism, which is used to learn from sections of sequences derived from MIDI scores, and subsequently replicate the data. LSTMs are a type of Recurrent Neural Network, an RNN looks like this:



Unfolded, it shows how every prediction at time t(h_t) is dependent on all previous predictions:

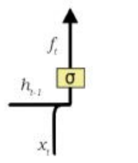


LTSM network improve on this model by using memory blocks, called cells. Two states: hidden and cell state can be transferred to the next cell:

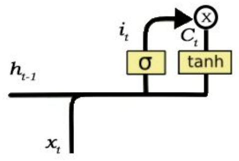


There are three types of gates: forget gates, input gates, and output gates. Forget gates remove information no longer required from the cell, and take two inputs: h_t-1 and x_t. Input gates add information to the cell states and use three steps to do so:

1) Regulating what values need to be added to the cell state by involving a sigmoid function. This is basically very similar to the forget gate and acts as a filter for all the information from h_t-1 and x_t.
2) Creating a vector containing all possible values that can be added (as perceived from h_t-1 and x_t) to the cell state. This is done using the tanh function, which outputs values from -1 to +1.
3) Multiplying the value of the regulatory filter (the sigmoid gate) to the created vector (the tanh function) and then adding this useful information to the cell state via addition operation.



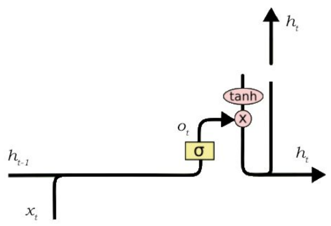
Forget gate



Input gate

The output gate uses useful information in the current cell and outputs the information. This is also done in three steps:

1. Creating a vector after applying **tanh** function to the cell state, thereby scaling the values to the range -1 to +1.
2. Making a filter using the values of h_t-1 and x_t, such that it can regulate the values that need to be output from the vector created above. This filter again employs a sigmoid function.
3. Multiplying the value of this regulatory filter to the vector created in step 1, and sending it out as a output and also to the hidden state of the next cell.



Output gate

2. Evaluation - Subjective Methods

2.1 Number of recommendations generated

To determine the number of recommendations generated, the performance of the *Neural Composer* system was evaluated. With the idea in mind that the top recommended songs would be converted to

MIDI files, and then used as input for training the RNN, the number of files used as input were considered.

To determine the amount of MIDI files input, six varying numbers of MIDI files were used as input. With each set number of input MIDI files, the model was run for 10 epochs (justified in section 2.4). The average time per epoch was calculated and plotted (Figure 1.8).

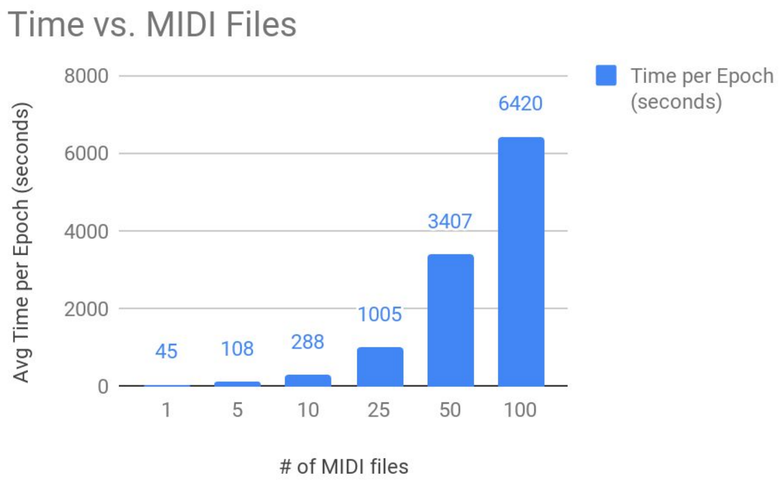


Figure 1.8 - Time vs MIDI Files

Taking into account the nature of most recommender systems (only the very small top n items are relevant), the two sizes we focused on were 5 and 10 MIDI files. The average time for 10 midi files to train per epoch was 288 seconds, and with 10 epochs run, the total time it would take to run would be 48 minutes. Comparatively, using 5 MIDI files, the total time to train the data using 10 epochs would be 18 minutes, which is 37.5% faster than using 10 MIDI files.

The next step was determining if using 10 files produced better results than 5 files. To do this, the system was run three times for each file size using the same genre of music. In this case, classical music was used due to its tendency of having well defined contrapuntal traits, which produce easily recognizable melodic and harmonic content. The results were subjectively measured by listening back to the MIDI files and determining if one produced more content than the other, and how similar that content was to the original MIDI files. Both sizes produced a similar amount of harmonic and melodic content, therefore it was decided to use 5 songs per experiment.

2.2 Length of generated MIDI files

In *Neural Composer* the length of the generated MIDI file is determined by the number of notes output. Initially 500 notes were output, amounting to approximately 2 minutes per file. This was determined to be unnecessarily long, and taking into account a 2017 study by Hubert Léveillé Gauvin, the average attention span for a piece of music presently is 5 seconds long compared to 20 seconds in 1980. Averaged out, this came to 15 seconds, which is ideal in ensuring users listen to the entire generated song. To output a file which produced around 15 seconds of music, the length of the sequence was set to 50 notes.

2.3 Learning rate

The learning rate determines how many notes are evaluated before a prediction is made. The set generated MIDI file length acts as a constraint to the maximum learning rate. As described in section 2.2, the length of the output was set to 50 notes, therefore setting the length to 50 or more notes would not allow the learning rate to take effect, and produce undesirable results. The following equation shows this, where $J(\theta)$ is the objective function, $\theta \varepsilon R^d$ and η is the learning rate

$$\theta = \theta - \eta * \nabla_\theta J(\theta)$$

If the learning rate (η) is too low, gradient descent will be slow. It it is too high, it may fail to converge or diverge, thus producing insufficient results by either overfitting or underfitting.

To determine the size of the learning rate, 10 epochs were run for six different learning rate sizes, ranging from 2 - 50. Each run used the same five MIDI files as input. The MIDI files used were of one genre, classical. Reasons for this decision are outlined in section 2.1. The results (Figure 1.9) show a learning rate set to 10 converging to a local minima at the fastest rate. Therefore a learning rate of 10 was used for the tests.

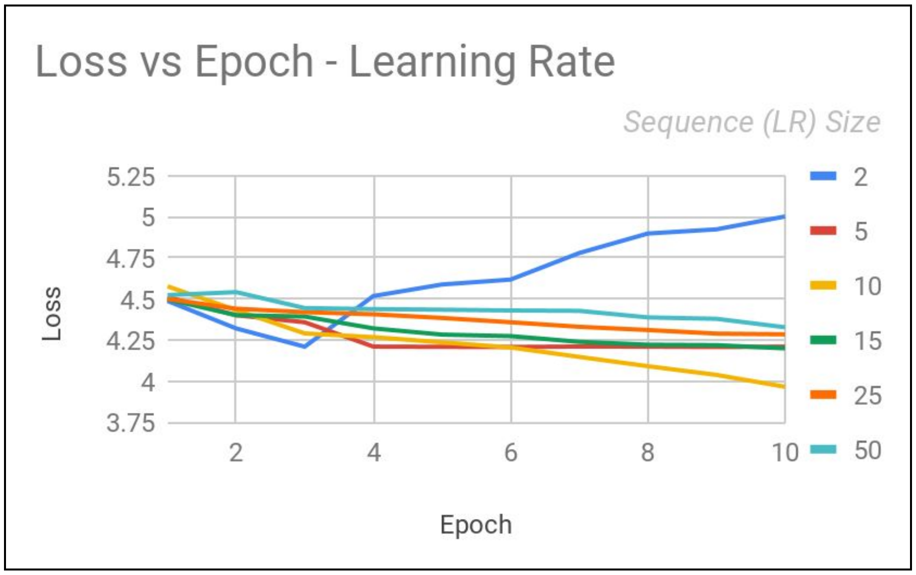


Figure 1.9 - Learning Rate

2.4 Number of epochs per test

The number of epochs was determined by test results of generated MIDI files. It was mostly subjective, taking into account the music content of the produced MIDI data. To determine this rate, the training was run on the set of five classical MIDI files for 30 epochs. This was done three times. The weights are saved for each epoch, every time the test was run, so all epochs before the 30th one were able to be used. Four epochs were sampled from each run: 5, 10, 15, and 30. After generating MIDI files for each epoch and respective run, it was determined 10 consistently produced results equally or more musical than the higher epochs. Therefore, to minimize testing time, it was decided to use 10 epochs per test.

3. Experiments and Analysis of Results

The experimental procedure consisted of loading six different combinations of songs into the *Neural Composer* and evaluating the results. Two songs were generated from the *Recommender System* and four were manually selected. Every song used in the experiments were instrumental, with no lyrical or vocal content. The table consisting of the songs used, the link to the audio file, and brief comments are shown in Figure 1.10.

3.1 Top songs - popular music

The first combination consisted of using the top 5 rated songs from one user, within one genre. This was generated by the *Recommender System.* Surprisingly, the generated music contained characteristics often found in popular music, with a polyrhythmic meter, repeating motives, and chord changes within the structure of the song. Although this contained one genre of music, popular music varies a significant amount in terms of style. As seen in the song list, it consists of rock, heavy metal, hip hop, and electronic. From the results, it can be inferred that although popular music ranges in terms of style, many songs within that realm contain similar traits.

PERSONALIZED MUSIC EXPLORER - Test Results			
Ratio (5 songs)	**Input**	**Link to Output**	**Comments**
Top songs from one user based on ratings: Popular Music	Secrets - One Republic Drop The World - Lil Wayne / Eminem Fireflies - Owl City Drops Of Jupiter - Train Beauty School (Album Version) - Deftones	https://drive.google.com/open?id=14KZs7WWP9cgfVT4pr2m-8CQ2rOORCjpn	Polyrhythmic and repeating motives Steadily changing triads, often used in popular music

Songs from one genre: Classical	**Classical:** Fantasia C major (Wanderer): Allegro - Schubert Prelude and Fugue in C major BWV 846 - Bach Prelude and Fugue in D major BWV 850 - Bach Prelude: Lento - Agitato - Rachmaninov Sonata No. 5 C minor: Prestissimo - Beethoven	https://drive.google.com/open?id=1KXaHKFj4beS8RABTgwM0a4c6FBvbs1iE	Repeating arpeggios structured similarly to many classical compositions, particularly Bach's.
Two drastically different genres	**Three Indian Ragas:** Adana Behag Durga **Heavy Metal:** Iron Maiden - Black Sabbath Crazy Train - Ozzy Osbourne	https://drive.google.com/open?id=16OjtdkHNI5bdo6q4qHpS7X77-Mutztn3	Moves between chromatic dissonance and vague melodic structures Monophonic melody, no polyphony
Genres commonly mixed (Classical and Rock)	**Classical**: Prelude and Fugue in C major BWV 846 - Bach Prelude and Fugue in D major BWV 850 - Bach Prelude: Lento - Agitato - Rachmaninov **Rock:** Smoke on the Water - Deep Purple London Calling - The Clash	https://drive.google.com/open?id=1Lj60OqEnnvhRSE1hZK0G-KvSpPCzn_oF	Arpeggios and scales outline changing chord structres. A mix between classical melodies and rock chord changes This is seen in many compositions by groups such as The Beatles and The Beach Boys
Randomly picked from all users	Unknown	https://drive.google.com/open?id=15iKTmw5nJPl001UtvyK0O5FilzWCYOck	Chords and various notes present, but with no discernable patterns
SVD Collaborative Filtering output for one user (serendipity)	Top 5: 1. You'll Be In My Heart - Phil Collins 2. The Maestro - Beastie Boys 3. Playground - Sia 4. Back To You - John Mayer 5. Undo - Björk	https://drive.google.com/open?id=1NDg7Yhl9y-8fdlsfbUfntOwaSPIcxXL1	Repeating melody throughout, with some changes. Out of the six tests, creates the most obvious melodic content

Figure 1.10 - Table of Results: All songs used were instrumental

3.2 Songs from one genre

The pieces from the selected genre, classical, were manually selected. This was done as a test to generate an expected result. Classical music has well-defined rules and limitations, and it was expected that the LSTM model would be able to learn and replicate these characteristics. For the most part, the generated music did in fact demonstrate this. The piece consisted of a repeated arpeggio, similar to that of Bach's "Well Tempered Clavier" compositions. A downside to the generated music, though, is that it continued to repeat without change for the remainder of the piece. This may be attributed to the fact that the piece is 15 seconds long, and if it had been longer, the trained sequence would be able to replicate additional harmonic change.

3.3 Two drastically different genres

The two manually-selected genres were three Indian ragas and two heavy metal songs. The purpose of this experiment was to attempt to create a new genre of music, similar to what has been done by artists like Lebanese singer Fairuz, who often blended modern Western and Arabic music. Using these two genres produced cacophonous results, though. The MIDI sequence goes back and forth between melodic content and dissonant, chromatic runs. The two genres use drastically different instruments that work with different scales. Heavy metal usually includes, a vocal melody, distorted guitar, bass, and drums, with the melodic and harmonic content consisting of traditional Western 12-tone scales. Contrarily, Indian Ragas can a have a variety of instrument arrangements, but often include sitar, sarod, tablas, and even violin and harmonium. These instruments can adhere to western scales, but also use microtonal notes, something void in western heavy metal music. The input MIDI files reflect these differences in their tonal composition.

The generated file only output one monophonic line played on one instrument, rather than using multiple instrument lines and harmonies to reflect the original data. This still gives insight into how hybrid music compositions are created, taking characteristics of various genres has the potential to create completely new, unfamiliar sounding music.

3.4 Genres commonly mixed

The purpose of this test was to create music similar to genres which commonly blend styles. The two manually-selected genres were rock and classical, which are often blended together. Artists like The Beatles, The Beach Boys, and Venetian Snares all have implemented classical music traits into their music. Although the generated music was not polyphonic, a trait common in both classical and rock, It was not expected that the file would produce such similar melodic content as actual songs which have used these two genres. The note sequence consists of partial scale runs, which have a sonority that brings to mind classical music, but in context, many melodies in rock use the same technique. It was surprising that the network was able to identify certain melodic sequences found in both genres and replicate them.

3.5 Random

These songs were blindly, randomly picked. The song metadata was not displayed, with the idea that there would be no bias when listening to the generated song. As expected, the generated piece had no structure. Surprisingly there were chords, but they were sparse, and were mixed with harmonic content outside of the key signature.

3.6 SVD (Serendipity)

Using the SVD Recommender System, the top five songs recommended to a user, with no additional constraints were used. The purpose of this experiment was to potentially produce serendipitous results. The five songs recommended varied slightly in genre, but had mostly similar musical traits, such as singer-songwriter and strong melodic content. The results produced a monophonic melody, absent of any dissonance. It also showed slight repetition with some variation in the melody. Subjectively this may have been the most melodic piece produced, mixing the right amount of variation, melody, and repetition to produce a well crafted piece of music.

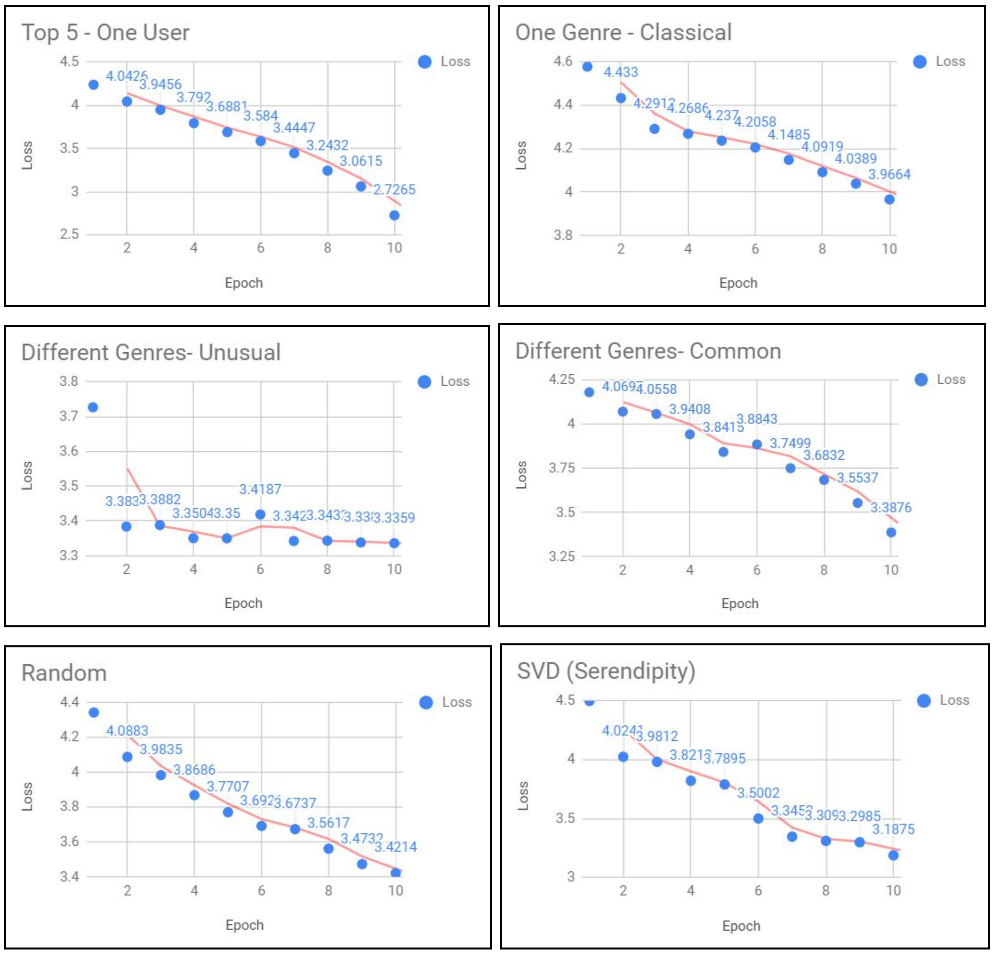


Figure 1.11 - Loss vs Epoch for each test

3.7 Loss vs Epoch

As seen in Figure 1.11, the loss functions with increasing epochs generated overall satisfying results. The Different Genres loss was somewhat unpredictable, but still managed to converge toward the local minima. This shows that perhaps the network was struggling to identify a predictable pattern.

Societal Impact

Legal and Ethical Considerations

Intellectual property must be considered in regard to this project, specifically copyright law. Copyright law concerns artistic expression and, unlike patents, Title 17 of the U.S. Code states that copyright covers anything that is functionless, or tangibly fixed [1]. In previous cases, MIDI files have been treated as sound recordings, which are covered by copyright because they are considered fixed. There has been dispute, however, about the fixed nature of MIDI recordings, because of the ability to easily edit the data.

In the current state of the Personalized Music Explorer, the MIDI files generated do not replicate the original copyrighted material enough to cause concerns of infringement. Currently running 10 epochs does not produce results that directly replicate the original input. Taking account for future improvements to the system, and training with more epochs, the generator module hyperparameters for the RNN and LSTM must be tuned in a such a way as to not overfit the input MIDI data. If the data is overfit enough to copy sections, or even entire songs, of copyrighted material, the system must be altered to avoid infringement. Ideally, implementing a feature which instills limits on the hyperparameters could resolve this legal risk.

Ethical considerations in this project focused mainly on the data used. For the Recommender System, using reasonably private user data was important. The project uses a subset of the Million Song Dataset called the "Taste Profile Subset" provided by music intelligence and data platform The Echo Nest. Upon release of the data, The Echo Nest released a statement regarding the privacy of the data, "the data in our Taste Profile release include a shuffled hash of persistent session identifiers from a very small random selection of our musical universe and only play counts associated with Echo Nest song IDs that overlap with the MSD set. There is no connection to individuals. The date added field is the date of the anonymized ingestion and is not the date of the original activity. No usernames, listener details, original IDs, dates, IPs, locations or anything but [random user string, EN song ID, play count] are being released."() The Recommender System is designed to exclusively use the data format provided by the Taste Profile Subset, ensuring no non-ethical data is used.

Contribution and Effects

The Personalized Music Explorer creates a tool to help users in multiple disciplines for a variety of purposes. For composers, it provides a way to create unexpected, serendipitous musical ideas, but it also allows them to use the musical content and arrange it for other ensembles. Musicologists and ethnomusicologists can use this system to identify patterns present in various genres and styles of music, and gather additional knowledge of musical traditions, cultures and trends. Similarly, this tool has the potential to aid music therapy research and brain science by deriving patterns that patients, whether suffering from Alzheimer's, or people on the Autism spectrum, deem to be effective.

Conclusions

Through generating music from song recommendations, the Personalized Music Explorer utilizes hybrid methods to generate serendipitous musical patterns. Taking advantage of the respective benefits of Collaborative and Content Based Filtering highlighted the benefits of using hybrid deep learning models. Through the experiments conducted, the foundation of potential applications was established, and in the process valuable musical patterns were identified, converged, and replicated. Insight was also gained about how the musical content affected the LSTM model. When using drastically different genres as input, the results indicated that the model had trouble extracting an identifiable pattern. As a result, the generated music was an unusual amalgamation of the two different genres, only using one instrument to represent the normally polyphonic and multitimbral traits found in both genres. Although challenging, this can be improved upon by modifying the model to incorporate different instruments and combine them discerningly. Also implementing a system which identifies each instrument part and outputs the part to the appropriate MIDI sound would greatly improve the quality of the generated music. Another improvement to this project would involve using Amazon MTurk to crowdsource a sufficient number of experimental results, allowing for a statistical analysis of the the results.

This project provided an opportunity to build off a multi-disciplinary cultivation of knowledge in order to create and explore the relationship between creativity and computer science. Machine learning and recommender systems both aim to gather new insight on existing aspects of life. Music composition in the context of these models allows users to gain, and build off, new comprehension of parameters present in all genres of music, and allows others to benefit from these discoveries.

References

[1] 17 U.S. Code § 1001 - Definitions. (n.d.). Retrieved June/July, 2018, from https://www.law.cornell.edu/uscode/text/17/1001

[2] Colin Raffel. "Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching". PhD Thesis, 2016.

[3] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.

[4] Pascanu, R., Mikolov, T., & Bengio, Y. (2012, November 21). On the difficulty of training Recurrent Neural Networks. Retrieved from https://arxiv.org/pdf/1211.5063.pdf

[5] Ricci, F. (n.d.). Item-to-Item Collaborative Filtering and Matrix Factorization. Retrieved from https://www.ics.uci.edu/~welling/teaching/CS77Bwinter12/presentations/course_Ricci/13-Item-to-Item-Matrix-CF.pdf

[6] Sarkar, Dipanjan, Analyzing Music Trends and Recommendations, (2017), GitHub Repository,https://github.com/dipanjanS/practical-machine-learning-with-python/tree/master/notebooks/Ch10_Analyzing_Music_Trends_and_Recommendations

[7] Shuai Zhang, Lina Yao, and Aixin Sun. 2017. Deep Learning based Recommender System: A Survey and New Perspectives.
ACM J. Comput. Cult. Herit. 1, 1, Article 35 (July 2017), 35 pages.

[8] Skúli, S. (2017, December 07). How to Generate Music using a LSTM Neural Network in Keras. Retrieved from https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5

[9] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere.
The Million Song Dataset. In Proceedings of the 12th International Society
for Music Information Retrieval Conference (ISMIR 2011), 2011.