Pop Music Hook Generation Using Neural Networks

Daniel Shen

Advisor: Professor Jeffrey Snyder

A Senior thesis submitted to the Music Department,
Princeton University in partial fulfillment of
the requirements for the degree of
Bachelor of Arts in Music

April 2022

I hereby declare that I am the sole author of this thesis.

I authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Abstract

While deep learning has recently gained popularity within a variety of disciplines in both industry and academia, applications of deep learning to music generation problems are comparatively less common. In this paper, I describe the process of designing from scratch a novel deep learning-based system for generating eight-bar-long pop hooks. I adopt an iterative approach where I design one model, analyze the merits and weaknesses of its output from both musical and technical perspectives, then use these to inform design decisions for subsequent models. My final model uses a recurrent neural network architecture and a generation strategy that feeds rhythmic "positional" encoding into the model along with standard event-based sequential data, which greatly improves its musical output. To verify that the model does not plagiarize, I rank the melodic similarity of the output to the rest of the dataset using a sequence similarity algorithm. Finally, I create an original dance-pop track, "Will You Learn", using a slightly modified version of an output example from my final model as the song's hook. The purpose of producing the song is to demonstrate the most likely use case for the model, which is to help kick-start musical inspiration for songwriters and/or producers. While my final model still has limitations, its musical quality is vastly improved from my first model and is at a level where an artist could potentially use it as a compositional tool.

Acknowledgements

First and foremost, I would like to thank the Korean boy group BTS for releasing the song "Dynamite" in 2020. Their uncannily slick pop sound was the original inspiration for this thesis, and without them it is unlikely that I would have ever undertaken the project.

Next, I would like to thank my advisor Professor Jeffrey Snyder for his guidance and support throughout the year. I would also like to thank Daniel Greenidge '18 and Byung-Cheol Cho '18 for their technical advice related to machine learning, a discipline that was new to me prior to this project.

Finally, I would like to thank the numerous friends and mentors I have had the privilege to meet during my time at Princeton for challenging me to grow personally and academically, both on and off campus.

Contents

1 Introduction

The growing popularity of deep learning as a discipline is well-documented in a variety of fields. These include image processing, natural language processing, product recommender systems, search engines, traffic prediction and navigation, and many other domains where a large quantity of data is either immediately available or able to be quickly collected. Compared to these "hot" domains, machine learning's impact within the realm of music composition has been relatively small, though this is rapidly changing as machine learning technology develops and relevant musical data becomes increasingly available.

Before proceeding, it is worthwhile to clarify some key terms. For the purposes of this paper, I define artificial intelligence (AI) as any form of intelligent behavior or decision-making exhibited by a computer. I define machine learning as the process of using large quantities of data to train artificial intelligence in computer systems. When describing fields of research, machine learning can be viewed as a subset of artificial intelligence. I define a neural network as a specific type of computational model for machine learning that utilizes an interconnected network of nodes that simulate biological neurons. Finally, I define deep learning as a subset of machine learning that uses neural networks, typically with multiple network layers (hence the term "deep").

Algorithmic music generation is not a new concept. The earliest example of a computer-generated musical score is "The Illiad Suite", created in 1959 by a team of researchers at University of Illinois at Urbana-Champaign [1]. Since then, there have been many landmark developments in computer-based sound synthesis and AI-based compositional systems. These developments are summarized in a wide collection of resources such as Roads's 1996 book *The Computer Music Tutorial* [2].

Despite the rich history of computer-based music generation, the specific application of deep learning systems to music generation tasks is still in its infancy. Briot

et al. [3] provide one of the first comprehensive surveys of deep learning techniques for music generation, yet their paper was published as recently as 2017. In their own words, Briot et al. were motivated to compose their survey due to "the lack ... of a comprehensive survey and analysis of this active research domain." Nevertheless, deep learning in music is rapidly gaining popularity. Even big names in industry have begun to participate. Google's Magenta Project [4], launched in 2016, encompasses a collection of AI-based tools for musicians and other artists, many of which are based on deep learning techniques. Since their acquisition of The Echo Nest in 2014, Spotify has featured machine learning as a core part of their R&D division [5] and has released publicly available academic publications as recently as February 2022 at the time of writing.

Previous works on deep learning-based music generation have spanned a wide range of objectives. These include lead sheet generation [6], generating variations on a theme [7], generation of chords/harmony given an input melody [8], MIDI performance generation (including note velocities and micro-timings) [9], and even raw audio waveform generation [10]. These works utilize a variety of popular deep learning architectures such as recurrent neural networks (RNNs), generative adversarial networks (GANs), convolutional neural networks (CNNs), variational autoencoders (VAEs), and restricted Boltzmann machines (RBMs), among others. Comprehensive summaries of these works can be found in the review articles by Briot et al. [3] and Ji et al. [11].

In this paper, I describe the process of designing from scratch a novel deep learning-based system for generating eight-bar-long pop hooks. The motivation for designing such a system is two-fold: the recent prolificity of deep learning as a discipline, and the relatively few (but increasingly frequent) attempts to use deep learning for music generation. In other words, deep learning is clearly a hot topic in 2022; might it have something valuable to provide for composers, songwriters, and/or producers?

As such, the aim of the project is practical: design a compositional tool that will help songwriters and/or producers to kickstart their creative process.

1.1 Defining the Problem Scope

I narrow down the scope of my music generation objective in three ways:

- *I only examine U.S. contemporary pop music.* Specifically, I only consider a subset of U.S. pop music with specific musical attributes. These attributes are more concretely described in section 2 when I discuss data preprocessing steps, but in short, the music must be from a corpus of U.S. pop, in 4/4 time, contain no changes in meter or key signature, feature a prominent and memorable melodic component, and possess a melody range that does not exceed two octaves. As will be explained in subsection 2.1, the primary motivation for selecting these attributes is to ensure a homogeneous dataset that will be easier for a deep learning model to train on. Of course, this means that many notable examples of U.S. pop are filtered out. In particular, all hip hop songs are filtered out unless their hooks contain sung melodies, a particularly significant loss given that hip hop is currently the #1 most popular sub-genre in U.S. pop according to some metrics [12]. Nevertheless, as section 2 explains, the number of pop songs that meet these criteria is still significant enough to construct a usable dataset.

- *I only output hooks rather than entire songs.* For this paper, I define a hook as an eight-bar self-contained unit of pop music. In pop, the hook often refers to a repeated refrain in the song and is often synonymous with the chorus. For this paper, however, I allow the hook to be any self-contained eight-bar segment of the song which can come from the intro, verse, chorus, bridge, or an instrumental break. In both music generation and natural language processing,

training a machine learning model to learn long-term structure is a non-trivial task [13]. The hypothesis is that focusing on outputting shorter musical chunks like hooks will be a simpler task for a machine learning model to learn and thus more likely to be successful. At the same time, hooks contain enough musical character to provide useful inspiration for artists; many famous pop songs are instantly recognizable by just whistling the tune of their intro or chorus. As such, they seem to be particularly high-impact generation objective to focus on. Ideally, then, the hooks would only be drawn from these high-impact "catchy" sections like intros and choruses. However, including verses and other sections allows me to construct a larger dataset, a particularly valuable asset for training a deep learning model. Issues regarding dataset size will be further explained in section 2.

- *I represent all hooks in a highly symbolic lead-sheet-like form.* As will be further detailed in section 2, I represent all hooks in terms of just two components: melody and chords. Because this representation bears much similarity to a lead sheet, I use pre-existing studies on lead sheet generation as a starting point for designing my model. In particular, I base many model design decisions off of the work presented by De Boom et al. who use RNNs[1] to generate lead sheets using the Wikifonia dataset[2], a collection of more than 6,500 lead sheets encompassing "all sorts of modern genres" [6]. Key ways in which I deviate from De Boom's model are the narrower scope of my dataset corpus (only pop songs with certain attributes) and the encoding scheme, the latter of which will be detailed further in sections 4 through 6. As for the rationale for using lead sheets, they are a compact of way of expressing fundamental musical information for a song in a format familiar to many musicians. This compactness reduces the number of

[1]For an introductory summary of RNNs, particularly in the context of music generation, see [3].
[2]Unfortunately as of April 2022, the Wikifonia dataset is no longer publicly available.

parameters a machine learning model would need to learn, thus increasing its chances of training successfully. At the same time, they encode enough musical information to be useful for a songwriter and/or producer while still allowing ample room for personal creativity; a jazz lead sheet, for example, is wildly re-interpretable depending on the musician behind the score.

1.2 Paper Outline

In section 2, I describe the dataset used as well as a number of simplifications I apply to the data as part of the preprocessing stage. In sections 3 through 6, I describe four generative models, each iterating on previous models. In each of these sections, I outline the encoding process, model architecture, and music generation strategy. I conclude each of these sections with an analysis of the merits and weaknesses of the model's musical output, which I use as a launching point for design changes for subsequent models up to my fourth and final model. In section 7, I place myself in the shoes of a songwriter/producer and describe the process of creating a full-length dance-pop track using a generated hook from my fourth model as initial inspiration. In section 8, I list several general insights that emerged from the process of designing a deep learning-based music generation model from scratch and provide suggestions for future research. I conclude my paper in section 9.

1.3 Source Code

For the models described in this paper, I use Google Colaboratory, a cloud-based Jupyter notebook manager, as my development environment. All source code used for this paper, including pre-trained serialized model files, can be accessed using the link provided in the appendix.

2 Dataset

I use the HookTheory TheoryTab database as a starting point for dataset construction [14]. HookTheory is a pedagogical online platform containing more than 30,000 user-submitted snippets of music formatted in a lead-sheet-like form (i.e. in terms of melody and chords). Due to copyright laws prohibiting uploads of full-length songs, HookTheory only contains song snippets which users assign labels such as intro, verse, chorus, bridge, and so on.

Using HookTheory for dataset construction proves advantageous for several reasons:

- *Music is already organized in terms of hooks.* The pre-partitioning of songs into musically meaningful sections such as verse and chorus fits well with my hook generation goal since a large number of these sections are roughly eight bars, or at least multiples of four.

- *Hooks are already in lead sheet form.* The song snippets are already formatted symbolically in terms of two streams (melody and chords) which is identical to the intended format of my generated output.

- *Database is up-to-date with respect to pop music.* The fact that HookTheory is an active online platform filled with user-generated content means that many of its song examples are from recent Billboard-charting pop hits. Examples of artists represented in the database include Dua Lipa, Ed Sheeran, BTS, Doja Cat, Ariana Grande, and Olivia Rodrigo, all of whom have singles in Billboard's 2021 Year-End Top 100 list [15]. This means that, assuming a well-constructed learning model, any potential generated output will be emulating currently trending pop music styles.

In order to ensure a reasonably homogeneous dataset, only hooks listed under the "pop" genre were considered. The JSON files for each hook were collected using a web

scraper and concatenated into a single master JSON file which was then serialized and imported into a Google Colaboratory development environment. A total of 4338 hooks matching the "pop" genre label were obtained.

2.1 Initial Preprocessing

After web scraping, the collected hooks were subjected to a series of preprocessing steps which affect all learning models subsequently described in this paper. This subsection contains a summary of these preprocessing steps, followed by a discussion on the rationale behind them from both musical and machine-learning perspectives.

First, an initial filter was applied:

- *Only consider hooks that stay in a single key, and standardize all keys to C.* In the majority of cases, this means C major or C minor, though other scalar modes such as mixolydian and dorian are also represented. Since HookTheory already encodes its hooks in terms of relative scale degrees, this process was fairly straightforward. However, all hooks containing key changes were removed.

- *Only consider hooks in 4/4 time.* All hooks not in 4/4 or containing multiple time signatures were removed.

- *Only consider hooks containing both melody and chords.* All empty hooks, or hooks containing only one or the other, were removed.

420 hooks failing to meet one or more of the above criteria were filtered out. From here, additional filtering and processing steps were applied in order to decrease the dimensionality of the data while still retaining essential musical information:

- *Only consider hooks whose melody range does not exceed two octaves, then normalize the melody range.* All hooks with melody ranges exceeding two octaves were removed, and the melodies of the remaining hooks were transposed

up/down until they fit between C3 and C6, resulting in 37 possible note values. The three-octave potential range accounts for the fact that some hooks may have a B as their lowest note or, similarly, a C♯ as their highest note. 122 hooks containing melodies exceeding two octaves were removed.

- *Ignore all rests that are in the middle of a melody or chord stream.* HookTheory encodes melody notes and chords in terms of events, where each event has an `isRest` boolean attribute. All note and chord events with `isRest` set to `true` were ignored. The musical effect is that each melody note and chord is sustained until either the beginning of the next note/chord or to the end of file. The only rests that were not ignored were those that occur at the beginning of a note or chord stream; each note or chord event also has a `beat` parameter denoting its time position, so streams with a beginning rest simply have a starting note/chord with a `beat` value greater than zero. For example, the chorus of "Someone Like You" by Adele has an eighth-note rest in the beginning of the melody; this rest was retained, though all subsequent rests in the melody were ignored.

- *Quantize all rhythms to a 16th-note time step grid.* All notes and chords that do not fall into the grid, such as eighth-note triplets, were rounded to the nearest 16th-note time step. Out of 186,130 total notes in the dataset at this point in the preprocessing stage, only 367, or less than 0.2%, do not fall into a 16th-note grid.

- *Simplify all chords to their root triadic forms.* This means that, for instance, all seventh chords were simplified to their triadic forms, and all inversions and slashes were ignored. Concretely, the chords were simplified to the following 31 possible values: `C-dim`, `C-maj`, `C-min`, `C♯-dim`, `C♯-maj`, `D-dim`, `D-maj`, `D-min`, `D♯-dim`, `D♯-maj`, `D♯-min`, `E-dim`, `E-maj`, `E-min`, `F-maj`, `F-min`, `F♯-dim`, `F♯-maj`,

`G-dim`, `G-maj`, `G-min`, `G♯-dim`, `G♯-maj`, `A-dim`, `A-maj`, `A-min`, `A♯-maj`, `A♯-min`, `B-dim`, `B-maj`, and `B-min`. Musical semantic differences between sharps and flats were ignored (i.e. D♯ and E♭ are considered equivalent). Only six augmented chords were discovered in the entire dataset, all of which were converted into major chords for simplicity. The reason that not every root has a diminished, major, and minor option is because some chords, such as `A♯-dim`, were found to never occur in the dataset. During this process, 106 hooks containing unusual and/or buggy chord definitions were discovered and removed, likely from legacy settings in the HookTheory platform.

- *Only consider hooks with bar lengths of four or multiples of eight, and standardize all lengths to be eight bars.* For hooks of length four, the hook was looped twice to create eight bars, and for lengths of multiples of eight, the hook was divided into eight-bar segments and each segment was treated as a separate hook. 242 hooks whose lengths did not meet this criteria were removed.

In total, 890 hooks were removed during preprocessing, yielding a sanitized dataset of 3448 hooks.

Musicians may understandably decry some of these sanitization steps as sucking the "life" out of music. However, many of these preprocessing steps are musically justifiable given the homogeneous nature of pop. Assuming a representative dataset, it appears that a sizeable fraction of U.S. contemporary pop music indeed stays in 4/4 time in a single key, possesses a melody range that does not exceed two octaves, can be seamlessly subdivided into a 16th-note time step grid, and contains sections whose bar lengths are multiples of four or eight, based on the small number of hooks that do not meet these criteria.

Removing rests and simplifying chords to their root triadic forms are harder to musically justify. The sustain length of notes and chords can have a profound impact on a hook's musical quality. Using 100% sustained legato melody notes in "Billie

Jean" by Michael Jackson, for example, removes much of its groove and punchiness. Sevenths and sus chords provide essential color that their root triadic counterparts lack, and ignoring slashes severely diminishes the musicality provided by some bass lines. Nevertheless, these simplifications are deemed necessary in order to reduce the number of parameters a training model would need to learn given the small size of the dataset.

Deep learning models typically require a large amount of data to train successfully. As Hadjeres [16] explains, one of the primary challenges for deep learning-based music generation models is the "trade-off between the size of the datasets and their coherence" given the scarcity of large publicly accessible musical databases and the complexity of music itself. That is, either one can select a large but heterogeneous dataset, such as all 30,000+ hooks on HookTheory or all 176,581 MIDI files in the Lakh MIDI dataset[3], or a small but homogeneous one, such as the 3448 sanitized pop hooks selected for this paper. In the former case, it is questionable whether or not a system can effectively learn and produce meaningful output from data that varies wildly in genre, instrumentation, meter, tempo, and other musical variables. In the latter case, it is difficult to train a system to generalize well based on a small number of training examples. For this paper, I choose to tackle the latter problem by working with musical data already represented in a highly symbolic form and applying the aforementioned preprocessing steps. Before a system can learn to decide whether a C, C/E, or Cmaj7 chord is appropriate at a given musical moment, it must first learn that C is more appropriate than C♯. With only 3448 hooks to learn from, the latter problem takes higher priority.

[3]The Lakh MIDI dataset was collated by Raffel [17] and, as of April 2022, can be accessed at https://colinraffel.com/projects/lmd/

3 Model 1: Simple Feedforward

For my first model, I emulate the MiniBach model described by Briot et al. [3]. MiniBach is a symbolic music generation system whose objective is to generate alto, tenor, and bass counterpoint for a given soprano input melody in the style of J. S. Bach's chorales. Its dataset is constructed using a corpus of 352 Bach chorales and splitting each into four-measure excerpts. Training is achieved using a single-step feedforward network and treated as a classic supervised learning problem: given an input four-bar soprano line, predict four bars' worth of alto, tenor and bass counterpoint. The temporal scope of the musical representation is global – that is, the system's prediction occurs in a single step, rather than note-by-note or beat-by-beat. The idea is to train the model using a corpus of Bach chorales so that it can generate convincing output given an arbitrary soprano melody as input. For more details about MiniBach's encoding scheme and construction, see [3].

Due to its simplicity, MiniBach has several important limitations:

1. *The output size is fixed.* Because of how the dataset and network are constructed, MiniBach can only generate exactly four bars' worth of music.

2. *Only accompaniments can be generated.* Generation depends on an input soprano melody. In its native form, this means that the final musical output is not 100% machine-generated. While the input soprano is arbitrary and can theoretically be algorithmically generated, this would necessitate a separate melody generation system.

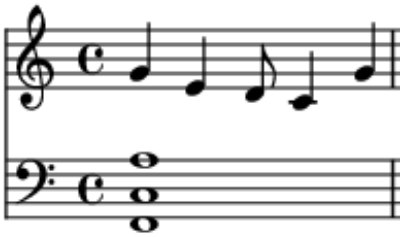
3. *Output is deterministic.* Given the same soprano melody, the model will always predict the same counterpoint.

For my model, I reframe the generation objective as follows: given an eight-bar input chord sequence, generate eight bars of melody. Since I already standardize

the length of each hook to be eight bars, limitation 1 poses no issues. Given the similarities of many chord progressions in contemporary pop music, I reason that the impact of limitation 2 will be smaller than expected, though it is still less than ideal. Later in this section, I describe a probabilistic solution for addressing limitation 3.

3.1 Encoding

I encode each melody note as a one-hot[4] vector of length 38 (37 possible notes after preprocessing, plus a hold symbol). Similarly, I encode each chord as a one-hot vector of length 32 (31 possible chords after preprocessing, plus hold). Each one-hot vector represents the identity of a note or chord at a given 16th-note time step, with holds used to represent sustain. Figure 1 provides an illustration of this encoding strategy.



```
hold    0 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1
C4      0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
D4      0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
E4      0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
F4      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
G4      1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

hold    0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
C-maj   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
F-maj   1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
G-maj   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 1: A simplified example of encoding a hook excerpt into one-hot vectors. The score on the left shows the first measure of the chorus of "Hey Soul Sister" by Train (obtained from the dataset). The diagram on the right shows its representation in terms of one-hot vectors. Each column represents a note and chord one-hot at a particular 16th-note time step. Consecutive hold notes denote the duration of sustain for the last non-hold note. In this simple schematic, the length of the note and chord one-hots are six and four respectively (including the hold note). The actual note and chord one-hot lengths are 38 and 32 respectively (including holds).

Using MiniBach as reference, I then proceed to encode an entire melody line for a given hook by flattening all one-hot note representations into a single vector. Since there are 128 16th-note time steps in a hook, this results in a one-dimensional

[4]For an explanation of the one-hot encoding scheme, particularly in the context of music encoding, see [3].

vector of length $38 \times 128 = 4864$. Similarly, to encode an entire chord sequence, I flatten all chord one-hots into a single vector, resulting in a one-dimensional length of $32 \times 128 = 4096$. For context, the flattened vector representations used in MiniBach for soprano and counterpoint are 1344 and 4480 respectively. I repeat this process for all hooks in the dataset, resulting in input (chord) and output (melody) arrays of dimensions 3448×4096 and 3448×4864 respectively.

3.2 Model Definition and Training

To define and train the neural network model, I use Keras [18], a high-level API built on top of the TensorFlow library developed by Google. The network architecture is shown in Figure 2. The input layer has 4096 nodes and the output has 4864, corresponding to the sizes of the vectorized chord and melody sequence representations described above.

The original MiniBach architecture has one hidden dense layer with 200 units, which I use as a starting point for designing my architecture. After experimenting with various hyperparameters such as number of hidden layers, sizes of each layer, and regularization parameters, I deviate from MiniBach slightly by including two hidden dense layers, each with 200 units and followed by a dropout layer with a dropout rate of 0.5, which gives marginally improved musical output over the single-layer architecture (namely, slightly less repetitive melody notes). As with MiniBach, the activation function for each hidden dense layer is ReLU, the output activation function is sigmoid, and the cost function used is binary cross-entropy. For the optimizer, I use Adam with a learning rate of 0.001[5]. Finally, I train the model on the dataset separated into input (chord) and output (melody) pairs for 20 epochs.

[5]For a technical explanation on the rationale behind these choices, see [3]. Because these types of parameters are commonly used in machine learning models, Keras has each built natively into its API and thus allows a user to implement them using minimal lines of code.

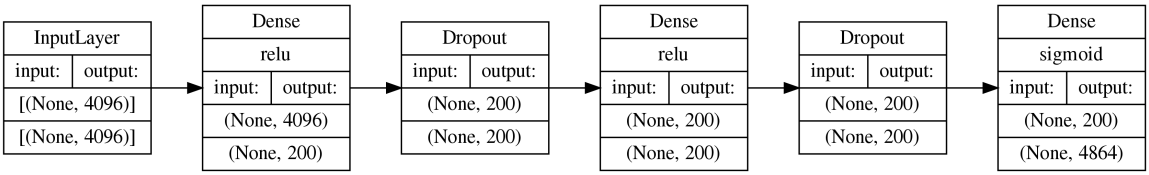


Figure 2: Network architecture for simple feedforward model.

3.3 Music Generation

My initial music generation strategy was to emulate MiniBach by feeding an input chord sequence, obtaining the predicted note probabilities for each time step, then choosing the note with highest probability at each time step. However, for all chord sequences tested, this strategy resulted in a predicted melody line entirely consisting of hold notes. This makes sense given two factors:

- Because of the melody encoding scheme, the majority of notes in the dataset are hold notes.
- Compared to Baroque choral music, pop melodies exhibit considerably more rhythmic variety.

Since MiniBach uses the same melody encoding scheme, the latter factor provides the most plausible reason for why this generation strategy succeeds for MiniBach but fails on a corpus of pop music. In Bach's chorales, nearly all quarter-note time steps contain note events due to the conventions of Baroque counterpoint. A system trained on a corpus of Bach chorales will therefore have a high probability of selecting a non-hold note at quarter-note intervals (i.e. time steps 0, 4, 8, 12, etc.) since non-hold notes frequently occur at these time steps. U.S. contemporary pop music does not exhibit such strict rules regarding note locations in rhythm space – many successful pop hooks, particularly those in dance or funk genres, rely on irregular syncopation to generate groove and energy – meaning the system is more likely to guess hold notes if the majority of notes are holds.

To circumvent this issue, I apply an additional constraint: if the predicted note is a hold, the system has a 25% chance of choosing the most probable non-hold note instead. This translates to approximately 25% of time steps being occupied by a non-hold note in the generated melody, or about four notes per measure, albeit with potentially irregular rhythms. Adding a stochastic element to the generation process also removes the model's deterministic limitation described earlier. This constraint will henceforth be referred to as the "25% rule".

Figure 3 shows an example score produced by this model using the chord sequence of the chorus of "Hey Soul Sister" by Train as input. Figure 4 shows another generated score, this time using the chord sequence of the chorus of a minor-key song, "LoveGame" by Lady Gaga, as input.

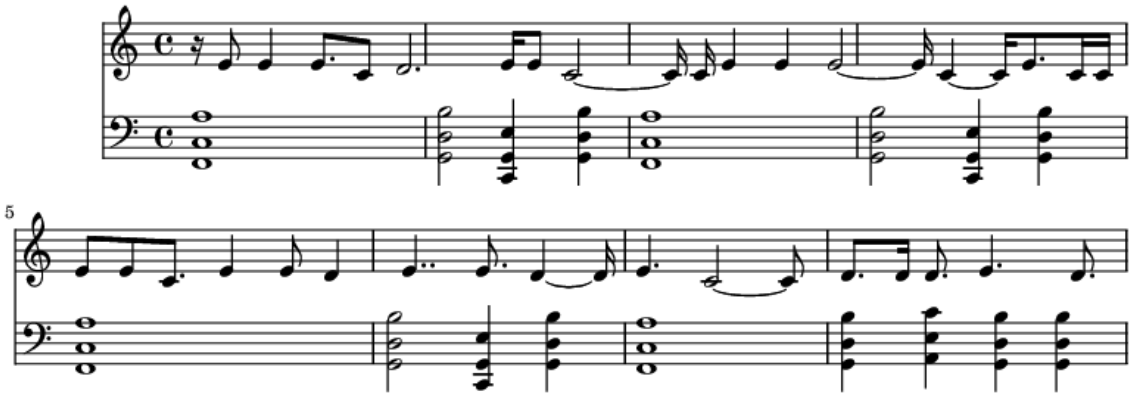
3.4 Analysis

3.4.1 Positive Attributes

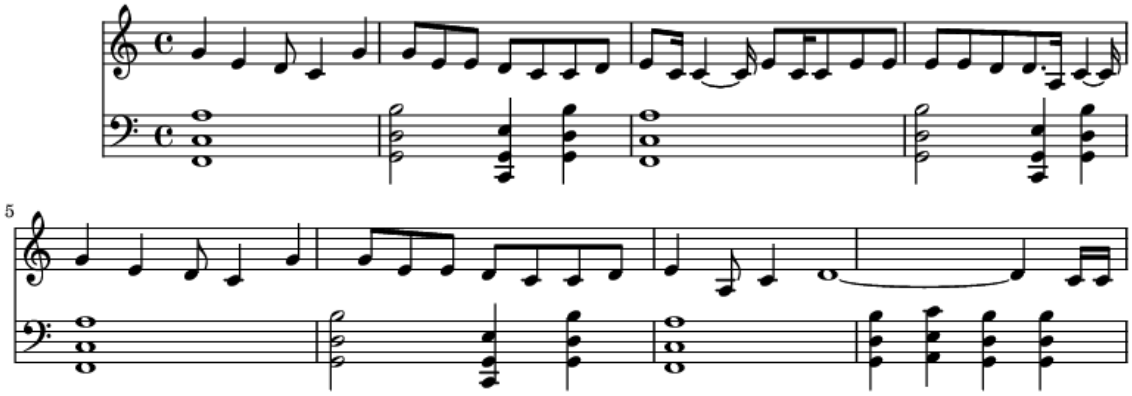
Several aspects of the generated output show promise and are worth highlighting:

1. *The generated melody contains mostly consonant notes.* All notes can be mapped to reasonable scalar modes that fit the input chord progression. The model appears to have learned, for instance, that F♯ is not a particularly likely note choice for a song in C major.

2. *The model is able to distinguish between major and minor modes.* The model tends to pick E for major-key songs such as "Hey Soul Sister", and it tends to pick E♭ for minor-key songs such as "LoveGame". This aspect is particularly remarkable since the model has no prior notion of major/minor keys – it can only see which melody notes tend to pair with which chords – meaning it appears to have independently learned how to distinguish between major and minor.



(a) Generated hook

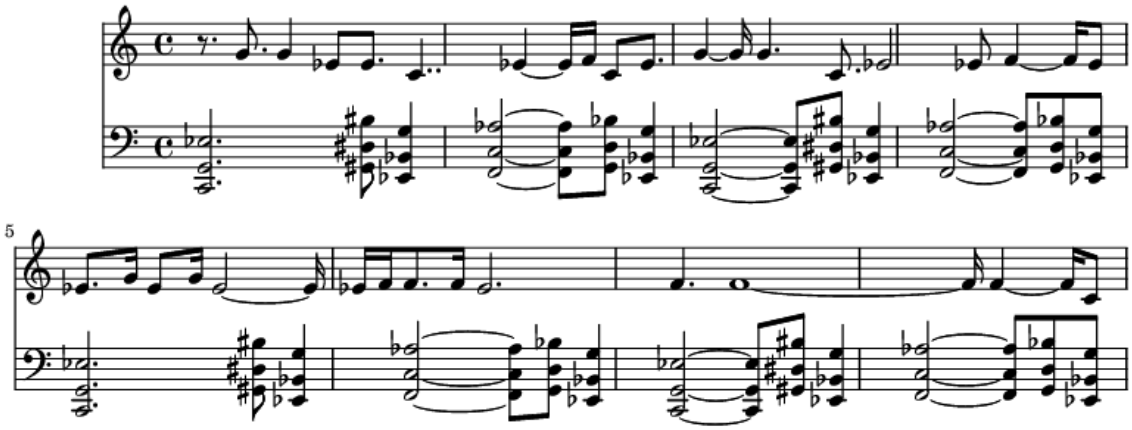


(b) Actual hook for reference

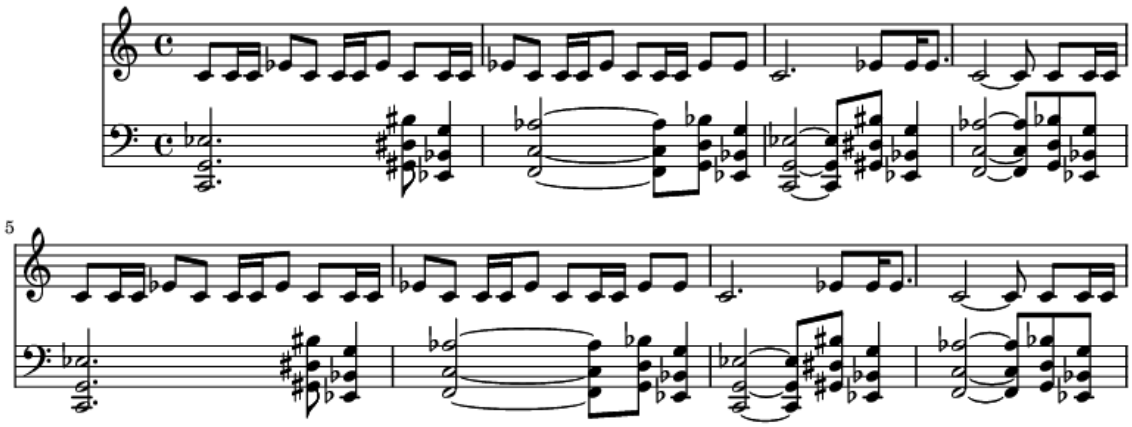
Figure 3: Generated hook using the simple feedforward model and the chord sequence of the chorus of "Hey Soul Sister" by Train as input. The actual chorus of "Hey Soul Sister" is also shown for reference.

3. *The generated melody's range is not too large.* With a handful of exceptions, most successful pop hooks possess relatively small melodic range (usually no more than one octave) since such melodies are easier to sing. The melodic range of the generated output rarely exceeds a fifth, the maximum range shown in figures 3 and 4.

4. *The model does not appear to plagiarize.* In other words, overfitting does not appear to be an issue at this point. The topic of plagiarism will be discussed in more detail in section 6 when describing my final model. For now, we can simply observe that a reasonable listener would not view the generated melodies



(a) Generated hook



(b) Actual hook for reference

Figure 4: Generated hook score using the simple feedforward model and the chord sequence of the chorus of "LoveGame" by Lady Gaga as input. The actual chorus of "LoveGame" is also shown for reference.

as plagiarized copies of the actual melodies.

3.4.2 Negative Attributes

Some key musical limitations of the model's output include the following:

1. *The model can only generate eight bars' worth of melodies.* As with MiniBach, the model needs to be fed an eight-bar input chord sequence in order to generate exactly eight bars' worth of melody. One possible way to generate melody and chords simultaneously would be to train a second model that predicts chords

given a melody line, then use the melodic output of the first model to generate chords with the second. This strategy is left as a possible subject for future research and is not investigated further in this paper.

2. *The rhythmic character of the generated melody exhibits little variance regardless of the input chord sequence.* In other words, the melody's rhythm always possesses the same semi-chaotic syncopation due to the 25% rule.

3. *The model tends to pick the same notes regardless of the input chord sequence.* It appears to heavily favor scale degrees 1 through 5, which are the most common scale degrees represented in the dataset.

4. *Although the notes themselves are consonant, the melody exhibits little sense of phrasing or direction.* In particular, there is no evidence of deliberate rhythmic repetition, mostly due to the indiscriminate nature of the 25% rule. The notes themselves also meander through the eight-bar phrase without a clear notion of melodic phrasing.

3.4.3 Discussion

From a technical standpoint, the most glaring issue of the simple feedforward model is the high dimensionality of the input and output representations (4096 and 4864 respectively) relative to the size of the dataset (3448 hooks). With so many parameters to learn from such a sparse dataset, it is unlikely that the system would be able to learn to generate melodies beyond a "smeared" average melody based on note frequencies. Indeed, initial generation attempts failed because the system would simply guess holds for every time step, the most frequently occurring "note" in the dataset. While the 25% rule provides a temporary fix, it is far from an ideal solution. The combined result of these limitations, particularly limitations 2 through 4, means that the generated melodies exhibit little variance in musical character no matter the input

chord sequence, making it a fairly lackluster tool for producers and/or songwriters to jump-start musical inspiration.

In the following section, I describe a strategy for addressing these four limitations using an RNN instead of a feedforward network.

4 Model 2: RNN with Time Step Encoding

RNNs are a popular neural network architecture for music generation, particularly for variable-length musical output. While their variable-length output capabilities are not strictly necessary for my generation objective (outputting eight-bar musical hooks), other features of RNNs provide promising ways of addressing the limitations of model 1:

- *Compared to a simple feedforward strategy, the dimensionality of the input and output is smaller and the effective size of the dataset is larger.* RNNs are trained to predict the next element in a sequence based on previous items in the sequence. In the context of music generation, this means that both training and generation occur note by note or time step by time step, rather than in one single step. Assuming we define a sequential item to be a time step, it takes much less information to encode a time step than it does to encode an entire hook, and the number of time steps in the dataset is orders of magnitude larger than the number of hooks (128 times greater in my case, since all hooks contain 128 time steps). For a small dataset, this feature is especially advantageous.

- *RNN-based models make it easier to generate both melody and chords simultaneously.* As De Boom et al. note, melody-chord coherence is itself a challenging problem given the complex interplay between harmony, melody and rhythm [6]. Nevertheless, the work reported by the authors provide a good example of how RNNs are a potentially fruitful strategy for tackling this problem.

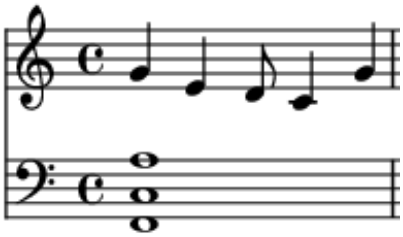
For my second model, I define the generation objective as follows: given an arbitrary input "seed" hook sequence, generate eight bars' worth of novel hook content in terms of both melody and chords. For now, I define a sequential element to be a 16th-note time step, meaning generation occurs time step by time step until 128 steps have been outputted. I generally use the work reported by De Boom et al. [6] as a starting point for encoding and architecture design decisions, with some key modifications that will be described below.

4.1 Encoding

To encode my dataset, I begin with the same one-hot vector encoding scheme described in section 3.1 and shown in figure 1. I retain the use of holds for notating note/chord sustain. However, I keep the one-hot vectors unflattened. That is, I represent a melody in terms of a 38×128 tensor instead of a 4864-long vector.

Next, I concatenate the note and chord one-hot vectors for each time step into a "stacked" representation and repeat this process for every hook in the dataset. Figure 5 illustrates a simplified version of this process. Finally, I prepare the dataset for RNN training by splitting the entire dataset into input and output sequences where an output sequence is offset by one time step relative to an input sequence. I set the sequence length for both input and output sequences to be 63 time steps. As a simplified example, if a hook contains time steps $\{a, b, c, d, e, f, g\}$ and the sequence length is 3, the input sequences would be $\{\{a, b, c\}, \{b, c, d\}, \{c, d, e\}, \{d, e, f\}\}$ and the output sequences would be $\{\{b, c, d\}, \{c, d, e\}, \{d, e, f\}, \{e, f, g\}\}$.

Using a time step-based encoding scheme and encoding notes and chords together for each time step, rather than an event-based scheme as proposed by De Boom et al. [6], enables me to circumvent a key limitation of the De Boom model, which is that chord changes are not allowed to occur independent of melody events. That is, in their encoding scheme, a hook is defined as a sequence of melody note events rather



step:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
hold	0	1	1	1	0	1	1	1	0	1	0	1	1	1	0	1
C4	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
D4	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
E4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
F4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
hold	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
C-maj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F-maj	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G-maj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5: A simplified example of encoding a hook into one-hot vectors, then concatenating the note and chord one-hots into "stacked" vectors for each time step. The score on the left shows the first measure of the chorus of "Hey Soul Sister" by Train. The diagram on the right shows the "stacked" vectorized encoding of the hook in terms of concatenated note and chord vectors. Each column represents a 16th-note time step. As in figure 1, the actual number of possible note and chord values is larger than what is shown in this figure.

than a sequence of time steps, and chords are only allowed to occur at the same time as a melody note; chord changes occurring below a sustained single melody note are not allowed. This means that certain hooks, particularly dance hooks with looping chord progressions such as the bridge of "Levels" by Avicii, are impossible to represent accurately. While this strategy results in a simple event-based encoding scheme that is easy to train, the musical restriction is significant. Using a time step-based approach offers a musically intuitive way to encode melody-chord independence while still retaining low dimensionality for inputs and outputs.

4.2 Model Definition and Training

I define an input layer which accepts inputs of size 63×70, since there are 63 time steps in each input sequence and each time step is represented by a vector of length 70 (38 possible notes + 32 possible chords). Using the De Boom model as a reference, I then feed this input into two LSTM layers, each with 64 units. The shape of the output logits are 63×70 which match that of the input layer. A schematic of the

network architecture is shown in figure 6. As with the simple feedforward model, I use the Adam optimizer with a learning rate of 0.001 and a binary cross-entropy cost function. Training is conducted for 10 epochs.

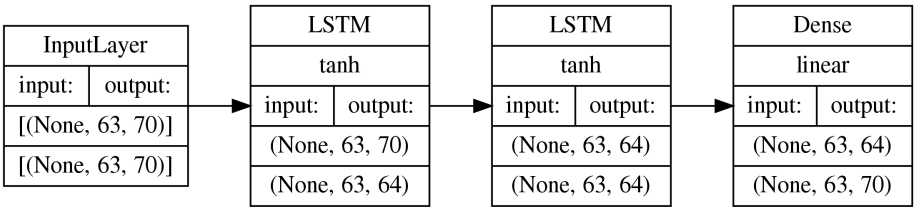


Figure 6: Network architecture for model 2.

4.3 Music Generation

To generate an eight-bar hook, I randomly select a 63-time-step-long input hook sequence from the dataset and feed the input into the model to obtain a 63-time-step-long output sequence, which represents the model's "prediction" of the next sequence element for every input time step. At this point, all "predictions" are still in the form of raw logits. I then take the *last* time step vector in the output sequence and apply the softmax function[6] with a temperature of 0.15 on the note and chord portions of the vector to obtain a probability distribution for the possible notes and chords at the time step. I then use `numpy.random.multinomial` to sample from these distributions to obtain usable note and chord predictions. I represent the sampled note and chord in terms of one-hots and concatenate them back into a time step vector, which I define to be the first time step of my generated output. Finally, I append this vector to the end of the input sequence, then shift the input window forward by one time step such that the window now includes the predicted vector at

[6] An introductory explanation of the softmax function can be found in [3]. Temperature can be conceptualized as a means of controlling how "confident" the resulting probability distribution will be. A higher temperature corresponds to less confident probabilities and a more uniform distribution. A lower temperature corresponds to a "harder" distribution with more confident guesses and more intense variation between probabilities. For this model, a temperature setting of 0.15 yielded the most musically pleasing results.

the end. This new input is then fed into the model, and the process repeats until 128 time steps have been generated.

Figure 7 provides a simplified schematic of the generation process. Figures 8 and 9 show examples of generated hooks using 63-time-step-long seed inputs from "Beneath Your Beautiful" by Labrinth and "Don't Stop Till You Get Enough" by Michael Jackson respectively.

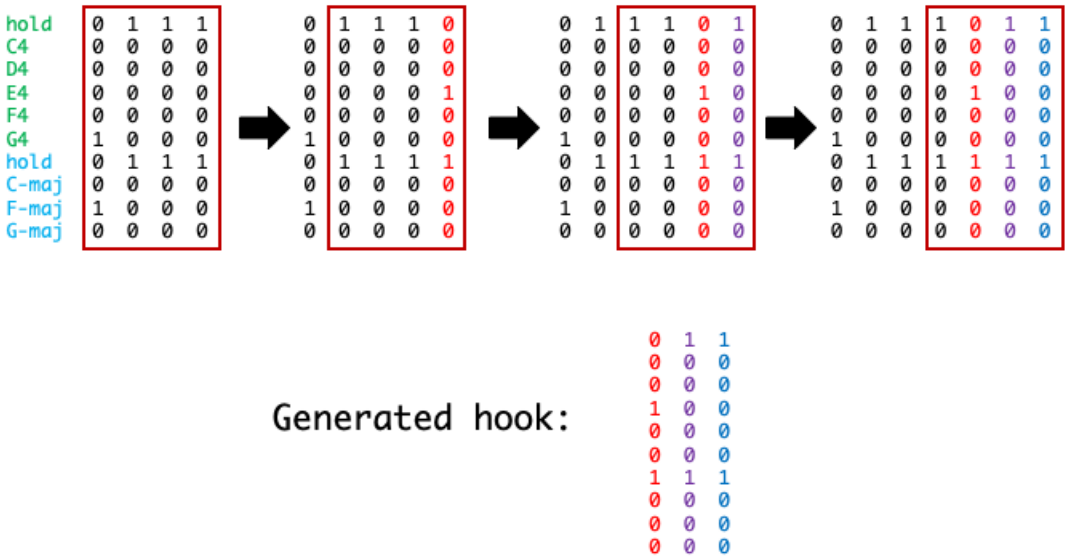
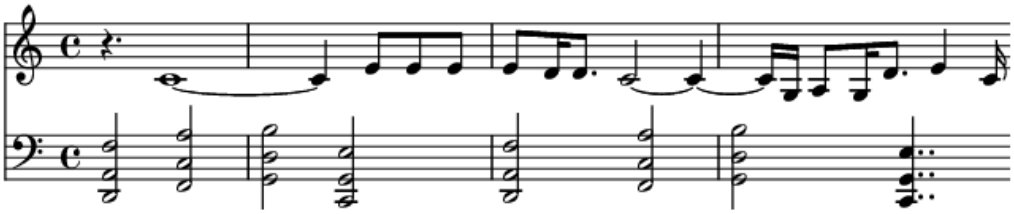


Figure 7: Simplified example of the music generation procedure for model 2 over 3 iterations. At each iteration, a predicted time step vector is obtained and appended to the input sequence. The input window is then moved forward by one time step and re-fed into the model to generate the next time step vector. The sequence of all predicted time step vectors defines the generated hook (shown at the bottom). In this schematic, the input window size is four and the time step vector length is 10. The actual window size and time step vector length are 63 and 70 respectively.

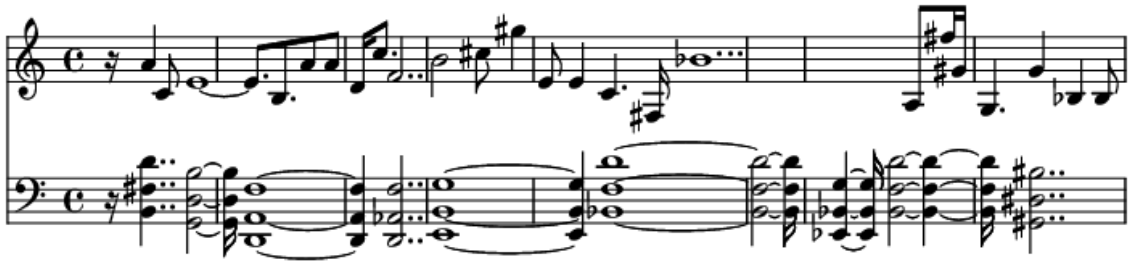
4.4 Analysis

4.4.1 Positive Attributes

Some musically promising aspects of model 2's output include the following:



(a) Seed input



(b) Generated hook

Figure 8: Generated hook using model 2 and a seed input from "Beneath Your Beautiful" by Labrinth.

1. *The rhythmic frequency of chords relative to melody notes appears to be plausible.* There are roughly four melody notes for every chord event, which is a musically reasonable amount.

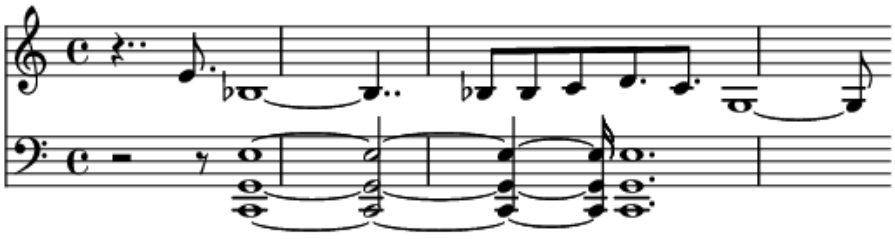
2. *The melody line appears to possess a rudimentary sense of phrasing and direction.* In the generated hook shown in figure 8, the melody appears to generally ascend in the first four measures before dropping down in measure 5.

3. *Compared to model 1, the generated output exhibits more musical variety depending on the seed input.* When fed a seed with fewer notes and chords, such as "Don't Stop Till You Get Enough," the generated output is sparser.

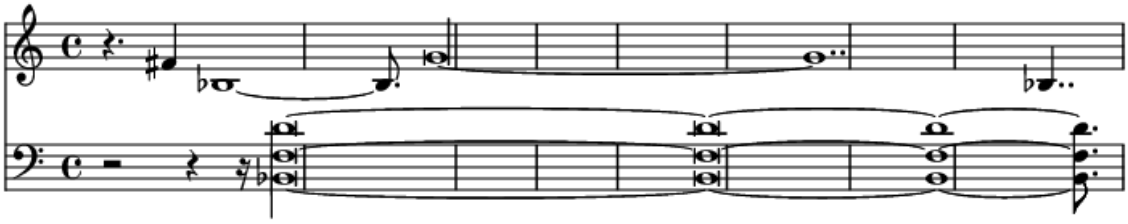
4.4.2 Negative Attributes

There are several glaring issues with model 2's output:

1. *The generated output is atonal.* There appears to be no consistent harmonic logic in the generated notes and chords. From the standpoint of pop music



(a) Seed input



(b) Generated hook

Figure 9: Generated hook using model 2 and a seed input from "Don't Stop Till You Get Enough" by Michael Jackson.

generation, which often exhibits simple harmonic grammar, this attribute is particularly condemning. It is also a regression from model 1 which at least outputs reasonably consonant notes.

2. *The generated output possesses little to no sense of rhythmic "place" within an eight-bar hook.* Because the 63-time-step input seed is randomly chosen from all possible 63-time-step-long sequence slices, the model often begins generation mid-hook. It is possible to circumvent this limitation by always choosing the last 63-time-step sequence of a hook as the input seed. However, this does not change the fact that in its current state, the RNN makes its next time step prediction only based on the previous 63 time steps and has no consideration of bar lines or meter. While it is theoretically possible for a system to learn these attributes if they are implicit in the sequence data, the model does not appear to have done so.

4.4.3 Discussion

Regarding issue 1, the most likely explanation is the use of hold notes to encode sustains, which also caused issues in model 1 and necessitated the implementation of the 25% rule. In the case of model 2, if the vast majority of notes and chords are holds, during the training stage the system can simply guess hold virtually every time and still be decently rewarded by the loss function. Because of the sparsity of non-hold notes and chords relative to holds and the small size of the dataset, the model is unable to learn the nuances of which non-hold notes to pick, and training for more epochs simply results in overfitting.

There are two possible solutions to address this issue. The first is to down-weight holds during the training stage, and the second is to use a different encoding scheme such as an event-based one used in the De Boom model. In the next section, I leave the former as a topic for future research and focus on the latter approach.

5 Model 3: RNN with Event-Based Encoding

For my third model, I use a similar RNN-based approach to model 2. The main difference is the use of an event-based encoding scheme which bears more resemblance to the De Boom model and which is described further in subsection 5.1. I define an event as either the beginning of a note, the beginning of a chord, or time step 0 for the few cases where a hook begins with a rest[7]. As will be explained in subsection 5.1, defining an event in this manner still allows me to encode melody-chord independence, unlike the De Boom model.

Before encoding, I apply an additional preprocessing step: *scan the dataset to obtain a distribution of all possible event durations, then throw out all hooks that do not strictly contain the top 16 most frequent durations.* I express event durations in

[7]Note that this means that time step 0 is always an event.

terms of number of 16th-note time steps. For example, a quarter-note-long event would have a duration of four. The purpose of this preprocessing step is to reduce the number of possible durations the model needs to consider (i.e. the dimensionality of the duration one-hot representation, to be explained further in subsection 5.1). Originally 46 possible event durations were discovered to exist in the dataset. While the decision to keep the top 16 is somewhat arbitrary, the 17th through 46th most common durations all have less than 100 occurrences, so 16 was deemed to be an acceptable cutoff that results in a reasonably low dimension while still retaining most hooks. 364 hooks were removed during this preprocessing stage, bringing the number of hooks in the dataset to 3084. In descending order of frequency, the 16 durations being considered by the model are 2, 1, 4, 3, 6, 8, 10, 16, 5, 12, 14, 7, 9, 18, 32, and 13.

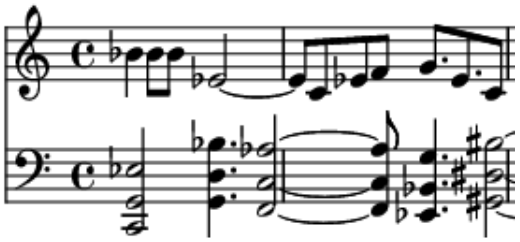
5.1 Encoding

I define a hook to be a sequence of events, where an event is the beginning of a note, the beginning of a chord, or time step 0. I encode each event in terms of three one-hot vectors, each encoding a note, chord and duration value. I then concatenate the three one-hots together into a single event vector, and repeat this process for all events in the hook. The note, chord, and duration values of each event are determined using the following algorithm:

- If the first time step has a hold (i.e. rest) in the melody stream, mark the first event's note as hold.
- If the first time step has a hold (i.e. rest) in the chord stream, mark the first event's chord as hold.
- If the next event is a new note but the chord is a hold, the previous chord is repeated.

- If the next event is a new chord but the note is a hold, the note is still marked as hold.

Figure 10 provides an example of this encoding scheme for the first 2 bars of the chorus of "I'm Coming Out" by Diana Ross. Retaining the hold note for both note and chord encodings in this manner makes it still possible to encode chord changes independent of melody notes. At the same time, the frequency of hold notes in both melody and chord streams is significantly less than in a time step-based scheme. The hypothesis is that a higher density of non-hold melody and chord values will allow model 3 to better discern which notes and chords to pick, at least compared to model 2.



i	n_i	c_i	d_i
1	B♭4	`C-min`	4
2	B♭4	`C-min`	2
3	B♭4	`C-min`	2
4	E♭4	`G-min`	6
5	hold	`F-min`	4
6	C4	`F-min`	2
7	E♭4	`F-min`	2
8	F4	`F-min`	2
9	G4	`E♭-maj`	3
10	E♭4	`E♭-maj`	3
11	C4	`A♭-maj`	2

Figure 10: Translation of the first two bars of the chorus of "I'm Coming Out" by Diana Ross into a sequence of events. The two-bar excerpt is represented as 11 events. Each event i is represented as a concatenation of three one-hot vectors encoding note, chord and duration. Note that chord values are repeated for as long as the chord is sustained. Note also that event 5 contains a new chord event over a sustained melody note, meaning the note is encoded as a hold. Finally, musicians may wince at the representation of an A♭ chord using G♯. This is an artifact of the LilyPond and Music21 score generation logic used for this paper and does not affect training in any way since note values are represented in terms of semitone integers.

One musical downside to this encoding strategy is that repeated chords are ignored. That is, the sustain length of a chord is now determined by the number of consecutive chords that share the same `semitone-class` identity, meaning any re-

peated chords will simply be "smeared" into a single long chord. This means that, for example, the dotted eighth chord rhythm for "Party Rock Anthem" by LMFAO can no longer be accurately represented.

There are three reasons why this additional loss of musical information is justifiable. First, such a simplification makes the final musical representation more similar to a traditional lead sheet which typically only contains non-repeated chords over a melody score. Second, not all HookTheory hooks contain accurate encodings of chord rhythms in the first place; due to its user-generated nature, some hooks contain elaborate rhythm patterns in the chord stream, while others opt for a more traditional lead sheet approach and only list sustained non-repeated chords. Third, as De Boom et al. explain for their model (which uses a very similar strategy), repeating chords in this fashion allows one "to model the entire lead sheet using a single shared time scale" [6]. In other words, as in model 2, a hook is defined in terms of a single sequential stream of events as opposed to two streams, making RNN-based training simpler.

To prepare sequential data for RNN training, I split the dataset into input and output pairs in a similar fashion as model 2. I use a sequence length of four, partly to accommodate the fact that the minimum number of events in a hook within my dataset is six. Later in section 6, I describe a way to increase the allowable sequence length while still taking into account hooks with small numbers of events, though these interventions do not apply for the current model.

5.2 Model Definition and Training

For model 3, I use a similar network architecture as model 2 but modify it to account for the new event-based encoding scheme. Figure 11 shows a schematic of the architecture. I define an input layer which accepts inputs of size 4×86, since there are four events in each input sequence and each event is represented by a vector of length 86 (38 possible notes + 32 possible chords + 16 possible durations). The input is

fed into two LSTM layers, each with 32 units. The output is a single event which represents the model's prediction of the next event given the previous four events[8]. As with previous models, I use Adam as my optimizer and a binary cross-entropy loss function. Training is conducted for 20 epochs.

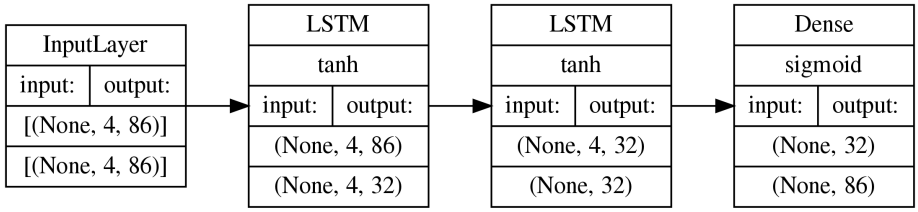


Figure 11: Network architecture for RNN model with event-based encoding strategy.

5.3 Music Generation

Generation is achieved using a similar strategy to model 2. The main difference is that rather than generating the next time step, the next event is generated instead. Note, chord, and durations are predicted by softmaxing the respective portions of the output logits, each with a temperature of 0.10[9], then sampling from the resulting distributions to obtain note, chord and duration one-hot indices.

Figures 12 and 13 show examples of generated hooks using four-event-long input seeds from "Over My Head (Cable Car)" by the Fray and "Hey Soul Sister" by Train, respectively.

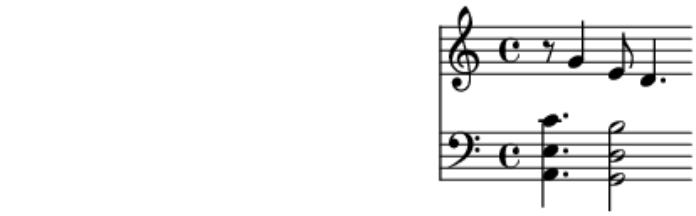
5.4 Analysis

5.4.1 Positive Attributes

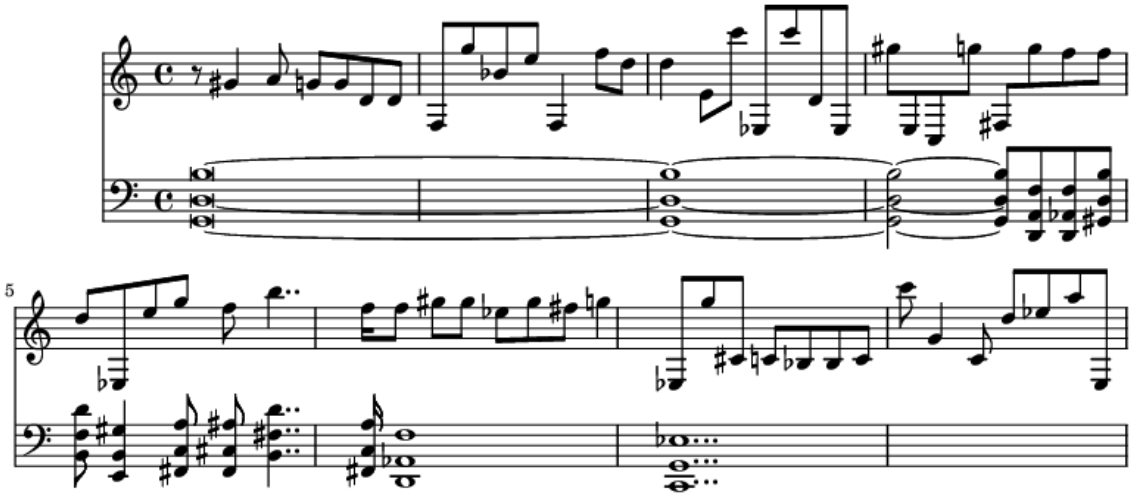
Musically promising features of the generated output include the following:

[8]The output representation differs slightly from that of model 2, which outputs not only the prediction of the next sequential item, but all "hidden" items as well. That is, in model 2, the output is a sequence of items rather than an individual item. This is a technicality based on how RNN layers can be defined in the Keras API that does not affect the training process.

[9]It is also possible to sample the note, chord and duration portions of the output logits vector at different temperatures, though this strategy is not explored in this paper.



(a) Seed input

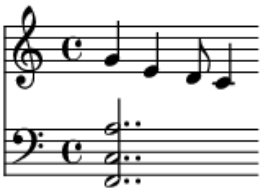


(b) Generated hook

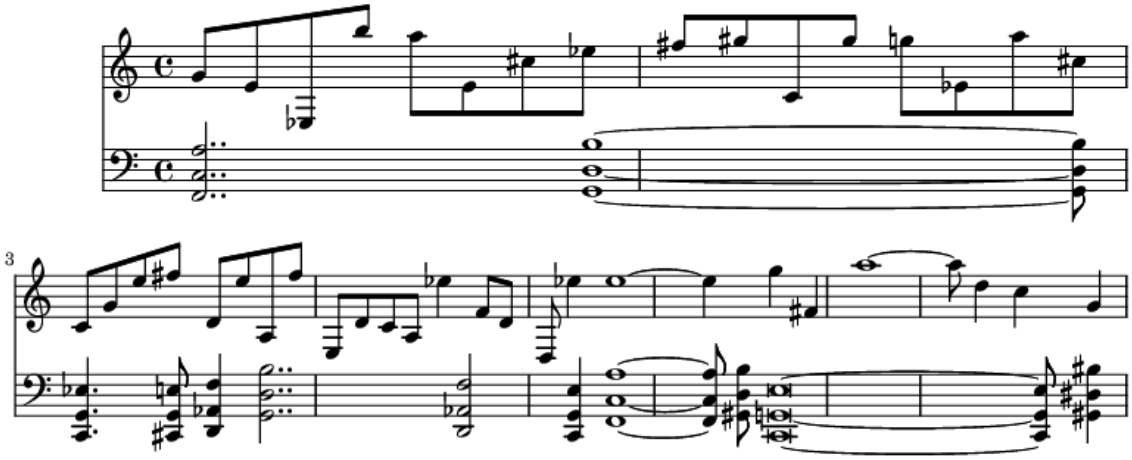
Figure 12: Generated hook using model 3 and a four-event-long seed input from "Over My Head (Cable Car)" by the Fray. The seed is from the middle of the chorus.

1. *Compared to model 2, the rhythms generated by model 3 are more realistic.* Instead of chaotic hyper-syncopated rhythms, there appear to be mostly eighth-note patterns in the melody with occasional variations using other common durations like quarter notes. Chords also generally line up cleanly with melody notes. Given the event-based encoding scheme, this is to be expected since the eighth note duration (two time steps) is the most common duration in the dataset.

2. *Most of the positive attributes of model 2 apply for model 3 as well.* Specifically, the rhythmic frequency of notes relative to chords appears plausible, and the melody appears to have a rudimentary sense of line.



(a) Seed input



(b) Generated hook

Figure 13: Generated hook using model 3 and a 4-event-long seed input from "Hey Soul Sister" by Train. The seed is from the beginning of the chorus.

5.4.2 Negative Attributes

However, there are still several key limitations:

1. *The musical quality of the output exhibits little variation regardless of the seed input.* This is most likely due to the short sequence length used (only four events) and indicates a regression from model 2. Intuitively, it is difficult to glean a sense of musical style from only four events' worth of music, so this outcome is to be expected.

2. *The output is still mostly atonal.* While there appears to be slight improvement over model 2, the musical quality is still at a level where no reasonable listener would mistake the output for an actual pop hook. This indicates that the model is still having trouble learning to choose harmonically reasonable notes

and chords despite the new event-based encoding.

3. *As with model 2, the generated model 3 still exhibits little to no sense of rhythmic "place" within the hook.* Since this limitation was not the primary focus of the design decisions for model 3, this is to be expected.

5.4.3 Discussion

Despite the aforementioned flaws, the significantly improved rhythmic quality of model 3's output, as well as the marginally improved harmonic quality over model 2, indicates that an RNN-based approach using an event-based encoding scheme shows promise. In the next section, I describe several modifications to model 3 designed to to address all three limitations listed above. These modifications are deemed substantial enough to warrant a new model classification.

6 Model 4: RNN with Event-Based Encoding and Additional Refinements

I begin my design of model 4 with three goals in mind, each meant to address one of the three limitations of model 3:

1. Modify the data encoding somehow so that larger RNN input sequence lengths are permitted.

2. Reduce the dimensionality of the note and chord representations even further to increase the density of note/chord data, making the model easier to train.

3. Implement some sort of internal "clock" within the model that will enable it to develop a sense of rhythmic place within a hook.

With these goals in mind, I apply the following modifications:

- *Remove all diminished chords from the dataset.* Given the small number of hooks that contain diminished chords, this simply means removing all hooks with diminished chords from the dataset. 167 additional hooks with diminished chords were removed, bringing the total number of hooks in the dataset to 2917. In doing so, the number of possible chords, i.e. the size of the one-hot vector representation for chords, is decreased from 32 to 22 (including holds).

- *Modulo all note values to fit into a one-octave range.* All melody notes are shifted up/down in octave increments until they fall between C4 and B4. While this severely modifies the musical quality of many melodies, particularly those that occupy the B-C break point, the reduction in note dimensionality is significant – from 38 to 13, holds included. At this stage, such a major simplification in musical representation is deemed warranted for the sake of training a successful model.

- *Encode an additional "position" vector with each event.* The position vector encodes the relative time step position of each event within either a one- or two-measure "chunk" of the hook. I elaborate on the details of the position vector's encoding and function in subsections 6.1 and 6.3.

- *If the user-specified sequence length is too long for a given hook, keep looping the hook until it matches or exceeds the sequence length.* For example, if the sequence length is set to 16 but a given hook only has six events, the hook is looped three times until it contains 18 events.

6.1 Encoding

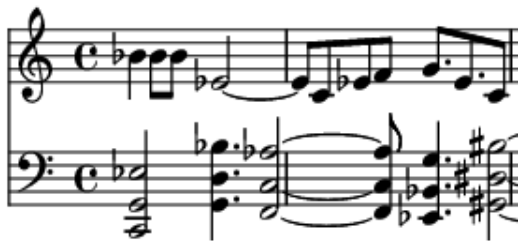
For encoding, I use a similar event-based scheme as model 3. I once again define an event as the beginning of a note, the beginning of a chord, or time step 0. I select note, chord and duration values for each event using the same algorithm described in

subsection 5.1. Figure 14 shows a schematic of this encoding scheme using the first 2 bars of "I'm Coming Out" by Diana Ross, the same example provided in subsection 5.1.

Despite similarities to model 3, there are several important differences. First, the one-hots used to encode note and chords are smaller in dimension (13 and 22 respectively, as opposed to 38 and 32). Second, and more importantly, there is now an additional "position" vector concatenated at the end of each event. The function of the position vector is to encode the relative 16th-note time step position of the event for every one- or two-measure "chunk" of the hook. For example, if the hook is split into one-measure chunks, the position value of the first beat of every measure would be zero, and this value would increment for every subsequent time step up to 15, after which it resets back to zero at the start of the next measure. If the hook is split into two-measure chunks instead, the beginning of every *other* measure has a position value of zero, and this value increments up to 31 before resetting back to zero. To clarify, two separate version of model 4 were developed: one where the position vector resets every 16 time steps, and one where it resets after every 32. To maximally reduce the dimensionality of the vector needed to encode this value while still maintaining binary encodings, I represent the position value as a four-bit binary value in the former case and a five-bit binary value in the latter[10]. These values are stored in vectors of length four and five respectively.

As to why only four- and five-bit versions of the position vector are considered for this paper, as opposed to six bits (which would represent four bars) or even seven bits (which would represent eight bars), the short answer is that increasing the number of bits also increases the chances of plagiarism. This will be more apparent once the music generation strategy is explained in section 6.3. At the same time, music in quadruple meter can already naturally be subdivided into multiples of four, so binary

[10]Special thanks to Byung-Cheol Cho, bccho@alumni.princeton.edu, for suggesting this idea.



i	n_i	c_i	d_i	p_i
1	B♭4	`C-min`	4	0 (0000)
2	B♭4	`C-min`	2	4 (0100)
3	B♭4	`C-min`	2	6 (0110)
4	E♭4	`G-min`	6	8 (1000)
5	hold	`F-min`	4	14 (1110)
6	C4	`F-min`	2	2 (0010)
7	E♭4	`F-min`	2	4 (0100)
8	F4	`F-min`	2	6 (0110)
9	G4	`E♭-maj`	3	8 (1000)
10	E♭4	`E♭-maj`	3	11 (1011)
11	C4	`A♭-maj`	2	14 (1110)

Figure 14: Translation of the first two bars of the chorus of "I'm Coming Out" by Diana Ross into a sequence of events using model 4's encoding scheme, which is based heavily on model 3's show in figure 10. Note the addition of the position vector p_i column. In this figure, the position counter resets after every one measure, or 16 time steps, so the position value is expressed as a four-bit binary vector. For the sake of readability for non-technical audiences, position values are shown both in decimal and binary, though only the binary representation is used during training and music generation.

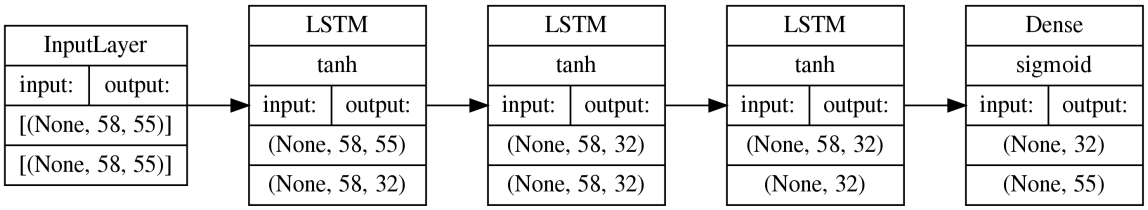
values offer a convenient way to represent the position counter in maximally compact binary vectors.

Finally, as with previous RNN-based models, I split the dataset into input and output sequences with a sequence length arbitrarily set to 58. This length is considerably larger than that used in model 3. To account for hooks with less than 58 events, I simply loop the hook until the number of events meets or exceeds 58.

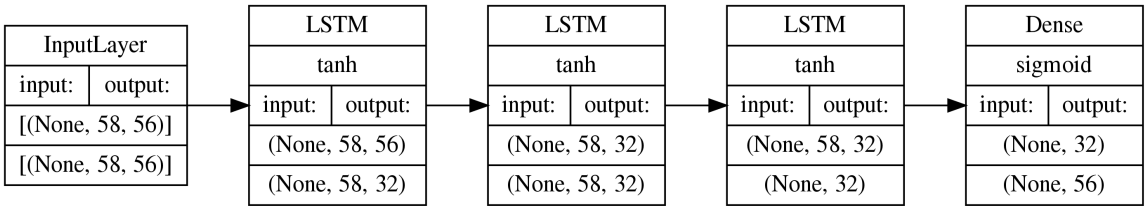
6.2 Model Definition and Training

For both four- and five-bit version of model 4, the network architecture is nearly identical to model 3. The only differences are the sequence length and the dimensions of the input and output layers which now account for the additional position vector. For the four-bit version, the input dimensions are 58×55 since there are 58 events in a sequence and each event is represented as a vector of length 55 (13 possible notes, 22 possible chords, 16 possible durations, and a four-bit binary position vector). For

the five-bit version, the dimension of the second axis is 56 instead of 55 to account for the extra bit in the position encoding. As with previous models, I use Adam as my optimizer and a binary cross-entropy loss function. In both four- and five-bit versions, training is conducted for 30 epochs.



(a) Four-bit version



(b) Five-bit version

Figure 15: Network architectures for both four- and five-bit versions of model 4.

6.3 Music Generation

Generation is achieved using the same "sliding window" strategy used in all previous RNN-based models, which is illustrated schematically in figure 7 (albeit with a time step-based approach that no longer applies). However, the position vector is treated differently from the note, chord and duration portions of the event encoding. As with model 3, note, chord, and duration values are predicted by softmaxing the respective portions of the output logits, then sampling the resulting distributions. However, the position portion of the output logits is ignored completely. Instead, the position value of a new event is determined by adding the position value of the *previous* event to the predicted duration of the current event, modulo the total number of possible position values. More succinctly:

$$p_i = (p_{i-1} + d_i) \mod 2^b$$

where p_i is the position value of the ith generated event, p_{i-1} is the position value of the event immediately *before* the ith event[11], d_i is the predicted duration value of the ith event, and b is the number of bits being used to encode the position vector (either four or five). As an example, suppose the model generates a new event and predicts its duration to be four after softmaxing and sampling. Suppose further that the position value of the previous event is 14, and we use a four-bit position vector encoding. Then the position value of the new event is $(14 + 4) \mod 16 = 2$. For the sake of easier conceptualization, I express all quantities as integers here, though in reality they are represented as either one-hot vectors (in the case of durations) or binary numbers (in the case of position values).

Once the position vector of an event has been calculated, it is incorporated into the event encoding vector along with the note, chord and duration predictions. As with previous RNN models, this event vector is then appended to both the generated hook and the end of the last input window, and the window is slid forward by one sequential element.

The effect of this strategy is that the RNN is fed "clock" information as well as standard note, chord and duration information as input for predicting the next event, with the position value functioning as the clock. The hypothesis is that having such clock information available will enable the model to make more musically reasonable guesses for note, chord and duration at every iteration. For example, if we assume that new chords tend to occur at the beginning of measures (a musically intuitive assumption for pop music), the model should now be able to pick up this pattern from the dataset. In other words, the model can now see *where* certain kinds of

[11]The previous event is either another generated event or, in the case of the very first generation step, the last event in the input seed.

events tend to occur in either a one- or two-measure chunk of the hook and should be able to replicate these patterns after training.

The reason why chunks greater than two measures long are not used in this paper is that longer chunks tend to increase plagiarism. When six- and seven-bit versions (corresponding to four- and eight-measure chunks) of model 4 were compiled and trained during the prototyping stage, the generated output exhibited clear plagiarism. The reason can be gleaned intuitively. In one case, the model sees that `F-maj` chords tend to occur at the beginning of a measure (position value 0 if using four-bit encoding, values 0 and 16 if using a five-bit). In the other case, the model sees that occasionally in the dataset, `F-maj` tends to happen precisely at time steps 0, 32, 64, and 96 (which is the case for "Hey Soul Sister" by Train). In short, encoding the exact time step position of all events in an eight-bar hook using a seven-bit binary value allows the model to uniquely identify a training sample. This means that during training, the model is more likely to simply learn how to predict the event sequences of specific songs rather than more general musical patterns. The challenge, then, is to select a chunk size that contains enough musical structure for the model to learn such patterns but not so large that it enables the model to uniquely fingerprint songs.

Finally, to address the issue of generation beginning mid-hook, I apply one more constraint: the first event of the generated hook must have a position value of zero. This means that the model will throw out all generated events until it generates one with a position value of zero, after which it will begin appending events until a full hook is generated. However, the thrown out events are still appended to the input seed and re-fed into the model as part of the "window"; they are just not allowed to be added to the hook until an event with position 0 has been generated at least once. For example, suppose the last event of the input seed is an eighth-note with position value 10, and we use a four-bit position encoding. If the RNN generates three more eighth-note events, the first two are simply ignored (though they are still

used as input into the model) and the third event, with position value 0, is now the first event of the generated hook.

Figures 16 and 17 show examples of generated hooks using model 4 and seed inputs from "Umbrella" by Rihanna and "A Sky Full Of Stars" by Coldplay respectively. In both examples, the four-bit version of model 4 is used. The four-bit version was perceived to have marginally improved musical output over the five-bit version. Specifically, it tends to generate slightly more harmonically reasonable melodies. However, the differences are both subtle and inconsistent. Because of this, the five-bit version is also included as part of the source code appendix, and the decision to only showcase two four-bit examples in this paper is somewhat arbitrary.

Note that in both of these examples, the displayed melody of the input seed only spans a one-octave range which reflects the new melody encoding.

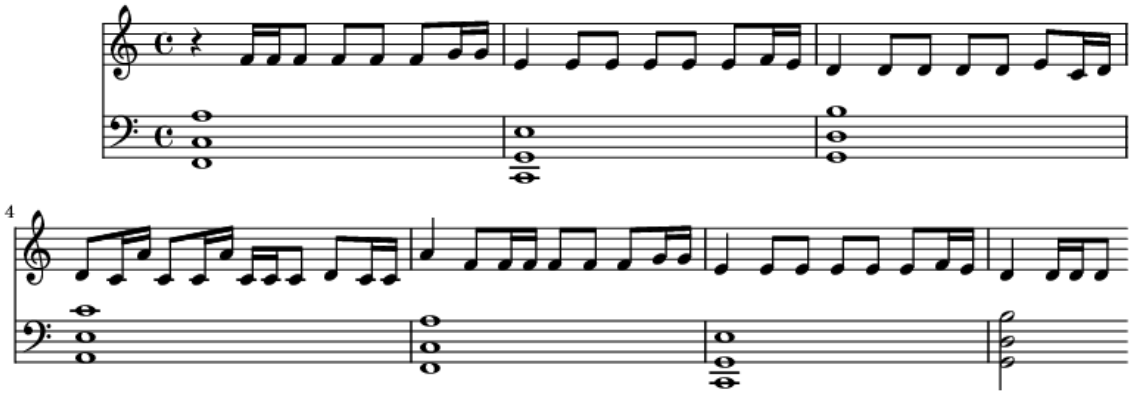
6.4 Analysis

6.4.1 Positive Attributes

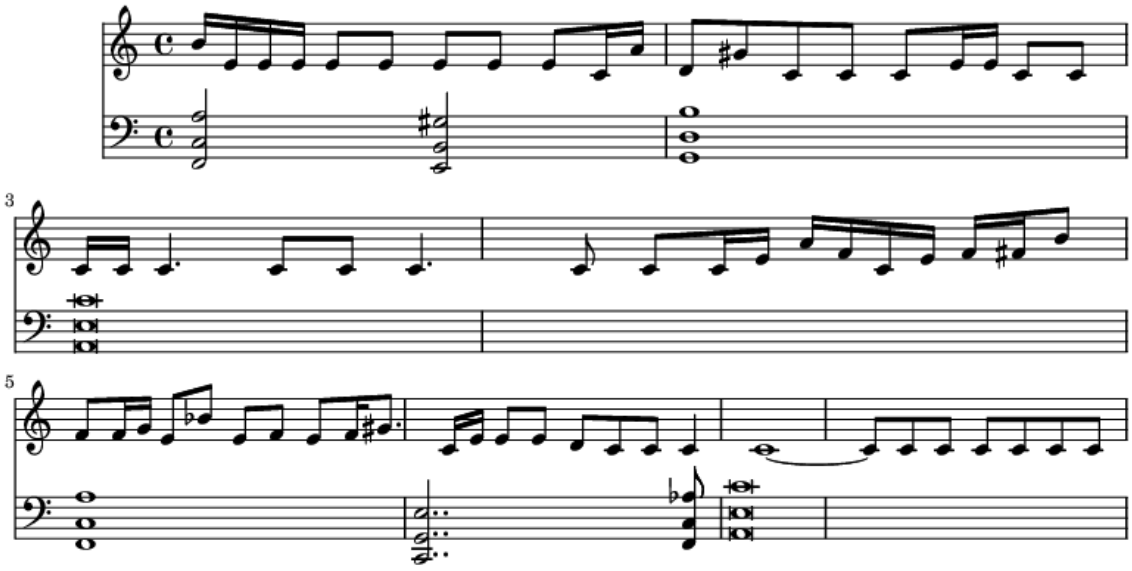
Compared to model 3, there are several noteworthy aspects of model 4's output that are worth celebrating:

1. *The rhythm is realistic.* Compared to the semi-chaotic syncopated patterns generated by models 1 and 2 or the monotonous eighth-note-dominated patterns of model 3, model 4's output exhibits a more human-like rhythmic quality. Notes exhibit some variation while still sticking to musically reasonable durations like quarter, eighth and 16th notes. The dotted eighth pattern in measure 5 of figure 17b, a classic pop rhythm, is especially encouraging.

2. *Chord changes are particularly convincing.* As expected, they tend to occur on the strong beats of measures (beats 1 and 3). The chords themselves also sound

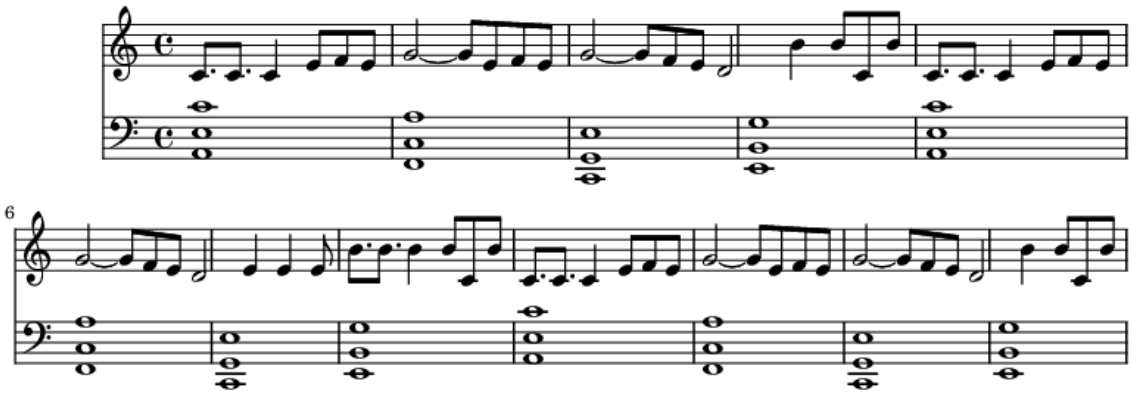


(a) Seed input

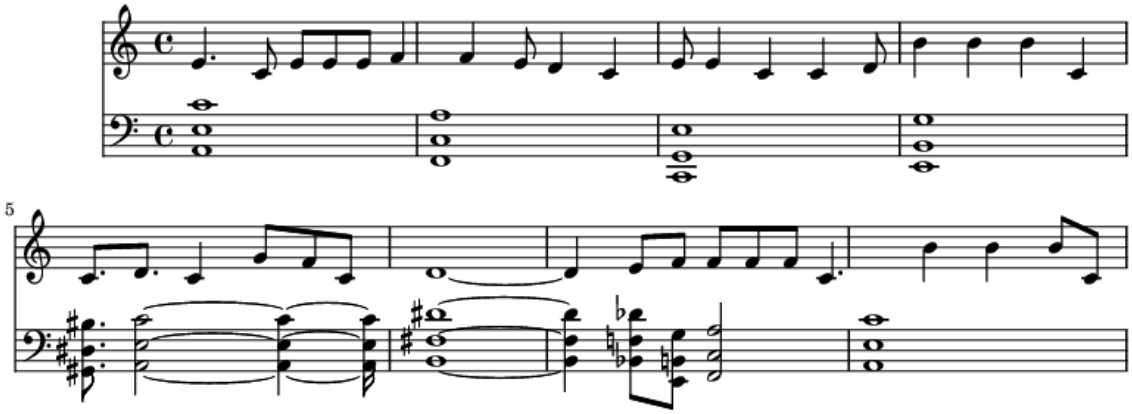


(b) Generated hook

Figure 16: Generated hook using the four-bit version of model 4 and a seed input from "Umbrella" by Rihanna.



(a) Seed input



(b) Generated hook

Figure 17: Generated hook using the four-bit version of model 4 and a seed input from "A Sky Full Of Stars" by Coldplay.

pop-like; most of the chords that occur on strong beats are common pop chords like `C-maj`, `G-maj`, `A-min` and `F-maj`.

3. *The harmony is significantly improved from model 3.* While the model still occasionally makes unusual note and chord predictions that border on atonal, most of the output sounds reasonably musical.

4. *The model appears to capture the harmonic and rhythmic essence of its seed input without plagiarizing.* In other words, the musical character of the output varies depending on the seed input, unlike models 1 and 3. At the same time, the model does not appear to simply plagiarize the seed hook. I verify this more rigorously using a custom plagiarism checker described in subsection 6.5.

5. *Occasionally, the model outputs legitimately catchy licks.* The first four bars of the generated output in figure 17b sound like a summer hit in the making, barring the awkward jump to B4 (which would likely be reinterpreted as B3 if a multi-octave melody range was used). In fact, it was this output example that motivated me to code a plagiarism checker due to how realistic it sounds; the concern was that the model may have simply been plagiarizing a different song with a similar chord progression. Fortunately, as will be shown in subsection 6.5, this is not the case.

6.4.2 Negative Attributes

Despite these improvements, there are still issues with model 4:

1. *The output tends to get "lost" as the hook progresses.* The musical quality of the first four bars tends to be higher than the second four bars. This is particularly evident in the example shown in figure 17. One possible reason is that as the sliding window iterative generation process keeps iterating, "mistakes" made by the model tend to accumulate. That is, bad guesses are re-fed into the model

as part of the input window, which results in a higher chance of making a bad guess for the next iteration, thus creating a positive feedback loop.

2. *The model still chooses atonal notes/chords somewhat regularly.* While their occurrences are less frequent than in models 2 and 3, they are regular enough to be distracting to a human listener. This is most likely due to the small size of the dataset which has been an ongoing issue for all models.

6.4.3 Discussion

Nevertheless, the results of model 4 exhibit substantial improvements from model 1. The quality of output, while imperfect, has now reached a level where a songwriter and/or producer could potentially use the model as a tool for jump-starting musical inspiration. All that he/she would need is an input seed (which can either be an existing pop hook or, theoretically, a user-created one), a reasonable tolerance for occasional atonal note and chord choices (which the user can clean up manually), and a willingness to manually "unfold" the one-octave range of the melodic output if necessary. Furthermore, because the system is RNN-based, the model can be easily tweaked to output musical lengths beyond eight bars if needed.

While model 4 has not yet reached a point where its output would be indistinguishable from a real pop song, the output exhibits enough novel musicality to provide a solid foundation for a human songwriter and/or producer to build on. In section 7, I put myself in the shoes of such a songwriter/producer and describe the process of taking an output example from the model (in this case, the example shown in figure 17b), making some tweaks, then turning it into a fully produced dance-pop track.

6.5 Plagiarism Checker

Due to the higher musical quality of model 4's output compared to previous models, a systematic way of checking for plagiarism is warranted. I approach plagiarism

checking as a sequence similarity problem and use the following strategy:

- Encode all melodies as sequential data.
- Use a sequence similarity algorithm to compare the melody in question against all other melodies in the database.
- Sort all melodies in the database by their similarity values as determined by the similarity algorithm, in descending order, then list the top n most similar hooks along with their musical scores and similarity values.

To encode a melody, I remove all durations and repeated notes and represent it as a sequence of integers between zero and 11, each representing a scale degree. For example, the first four measures of the chorus of "Hey Soul Sister" by Train would be encoded as $[7, 4, 2, 0, 7, 4, 2, 0, 2, 4, 0, 4, 0, 4, 2, 9, 0]$. Note the removal of all note repeats and the absence of any rhythmic information. I then use Python's `SequenceMatcher`[12] function to compare the melody in question against all melodies in the database. Finally, I display the top three most similar melodies along with their scores. For the "Umbrella"-based generated hook shown in figure 16, the most similar hook in the dataset is from "Say You'll Be There" by the Spice Girls with a score of 57.6%. For the "A Sky Full Of Stars"-based generated hook shown in figure 17, the most similar hook is from "Closer" by The Chainsmokers with a score of 70%. Full plagiarism check results for these hooks, including musical scores of the top three most similar hooks, are shown in the appendix.

It is worth mentioning that musical plagiarism is a complex and sensitive issue with potentially high monetary stakes and a colorful litigation history. The challenging question of what constitutes musical plagiarism in the first place, particularly in pop music with generally simpler harmony and structure, is beyond the scope of this

[12]For an explanation of the `SequenceMatcher` algorithm, see the Python `difflib` documentation at https://docs.python.org/3/library/difflib.html

paper, though algorithmic methods are often used as part of the analysis in court cases [19]. For the purposes of this paper, the sequence similarity method described above is sufficient for detecting egregious cases of melodic plagiarism, none of which were found for the generated hooks in question.

As to why only melodies are considered, melody plagiarism is generally more noticeable and artistically problematic than chord plagiarism, especially given that many pop songs share similar, if not identical, chord progressions (such as the classic I-V-vi-IV progression). To put it another way, if two songs have the same melody but different chords, one could possibly argue that one song has plagiarized another. This argument is harder to make if two songs have the same chords but different melodies.

7 Song Example

For my song example, I use the generated hook shown in figure 17b as a starting point. Since the first four measures sound more musical than the last four, I scrap the last four measures and loop the first four measures twice instead. I then tweak the last measure of each four-measure chunk slightly to create my eight-bar hook.

I choose mid-2010s pop EDM as my genre to showcase the hook, since this genre often features catchy hooks as a core part of song arrangement (typically during the drop). In the spirit of good humor, I choose lyrics that feature several generic phrases representative of the genre like "tonight" and "baby", though I trust that other artists will be more creative with their lyrical choices. For the verse melodies, I choose to emphasize the tonic C in an overly repetitive manner. This is mostly inspired by an intermediate stage during the development of models 3 and 4 in which selecting a softmax temperature that is too low (0.1 or lower) results in the system outputting a repetitive string of C notes. It turns out that this type of monotone melody fits reasonably well for my song verses. A link to listen to the song can be found in the

appendix.

8 Discussion

8.1 Insights

Below I list several insights that emerged through the process of iteratively designing a pop hook generation model from scratch as documented in this paper. The intention is for these reflections to be useful advice for machine learning researchers interested in diving into music generation or computer-science-inclined musicians interested in trying out machine learning for the first time.

- *When working with complex data like music, data encoding is the hardest part.* A large fraction of this paper is dedicated to describing the encoding process. Section 2 outlines musical simplifications applied during preprocessing, and sections 3 through 6 each contain an encoding subsection. In terms of time spending coding and debugging, the most time consuming part was by far the encoding process.

- *Having musical intuition makes encoding easier.* Having a musical background, or at least a baseline knowledge of music theory, makes it easier to discern which simplifications are worth applying and which are musically crippling. In section 2, I describe the musical rationale behind selecting the HookTheory dataset and the subsequent preprocessing steps. I ensure that at each stage, the loss in musical information is either not too severe or otherwise justifiable in some other way. This process relies heavily on pre-existing musical intuition and would be difficult to conduct for a non-musician.

- *Finding a suitable dataset can be challenging.* Compared to other popular targets for machine learning research, such as image processing or natural language

processing, the number of publicly available music datasets is slim and the quality of these datasets is often questionable. As explained by Hadjeres [16], there is often a trade-off between the size of the dataset and its quality (both in terms of raw musical quality and in homogeneity, an important factor for training machine learning models). Until the amount of publicly available musical data increases, future researchers will likely need to wrestle with this trade-off.

- *Neural networks are impressively robust learning models.* The fact that even model 1 is able to distinguish between major and minor keys is impressive and a testament to the powerful generalized learning capabilities of neural networks.

- *For music, the process of evaluating a model's output will necessarily be more qualitative.* This point may be particularly difficult for experienced data scientists to accept, since quantitative metrics are overwhelmingly the encouraged norm in scientific literature and qualitative evaluations such as harmonies "sounding less atonal" are more ill-defined. Unfortunately, the complexity of music (or any other artistic medium for that matter) makes it extremely difficult, if not impossible, to quantitatively measure musical character given that artistic interpretation is heavily subjective. In many cases, researchers simply take a survey-based approach and ask a sample of human listeners (often musicians) to rate the musical quality of a mix of real-world and machine-generated musical examples in Turing-test-esque fashion. De Boom et al. [6], Roy et al. [7], and Yeh et al. [8] all opt for such a survey-based approach. Of course, using survey data comes with its own risks such as bias in the sample population and biased assumptions in experimental design choices. In this paper, I use the compositional process for my song example as an alternative evaluation metric since it provides a more accurate simulation of how a music generation model may actually be used by the general public, though this is admittedly still highly

subjective. In short, researchers looking to engage in music generation models should expect to wrestle with more imprecise metrics for gauging progress.

- *Because neural networks are intrinsically difficult to understand, they are more difficult to tweak.* Even among experienced data scientists, neural networks are often viewed as "black boxes" [20] due to their complexity (often hundreds or even thousands of nodes in the case of deep learning) and the relatively autonomous process by which they are trained. While the researcher has choices over how the data is encoded, how the architecture is designed, how hyperparameters are tuned and which optimization protocol to use, the actual training process amounts to little more than executing code for some amount of time, tracking the calculated loss over time to make sure it is decreasing, and otherwise hoping for the best. Combined with the lack of clear quantitative metrics for gauging musical quality, this "black box" approach makes it more difficult to tune specific aspects of a model's output.

8.2 Future Work

While model 4 exhibits significant improvement over my first model, it is by no means a finished product. Furthermore, there are several directions a future researcher can take based on the work described in this paper beyond just improving the output quality of model 4. Below I list several examples of ideas for future work, in roughly descending order of priority.

- *Increase the size of the dataset.* This can be achieved by either collating more hooks from HookTheory or sourcing music from other databases such as the Lakh MIDI database or Musescore's public collection of lead sheets[13]. In the former case, the challenge would be to account for the fact that not all hooks on

[13] An official web-scraped collection of Musescore lead sheets can be accessed (as of April 2022) at https://github.com/Xmader/musescore-dataset

HookTheory are pop hooks. Either one would need to handle multiple genres or somehow ensure that the added hooks maintain a similar level of homogeneity as the ones used in this paper. In the latter case, the challenge would also be to account for variation in genre or even unspecified genres. Furthermore, one would need to account for differences in musical encoding and ensure a standardized representation. For example, the Lakh MIDI dataset only contains MIDI data; to represent chords as distinct entities, one would need to write logic to convert MIDI events into chords, which is not a trivial task.

- *Make additional refinements to model 4.* To recap, one issue with model 4 is that the musical quality of the last four measures tends to be worse than the first four. In subsection 6.4, I describe a possible reason: having a bad guess re-fed into the RNN model increases the chances of making another bad guess for the next iteration, creating a positive feedback loop. One potential solution is to modify the "sliding window" strategy so that generated events are not re-fed into the model; the window could simply loop continuously over the input sequence instead. The downside is that the creative potential of the generator would be severely diminished, and the chances of plagiarism would be higher.

- *Try a different architecture.* In this paper, I focus on feedforward and RNN-based architectures. Others, such as generative adversarial networks, convolutional neural networks, and variational autoencoders, among others, have shown promise in the literature [11], though the scope of the generation objectives may be different.

- *Package the model into a user-friendly app interface.* While the source code can be viewed and executed using the link provided in the appendix, the execution still occurs in Google Colab Jupyter notebooks which are difficult to navigate for a non-technical user. The notebooks also contain code at a level of detail

that is both overwhelming and unnecessary for a casual user to understand. Packaging the model as the back-end of a web app would greatly increase its utility for its intended audience, which is songwriters and producers, not data scientists.

- *Provide user-selectable options for the type of output desired.* In this paper, I only generate eight-bar samples of pop music without regard to the type of section being generated (e.g. intro, verse, bridge, chorus). Future iterations of this project ideally would provide the user the option to specifically output an intro, verse, chorus, etc. In addition, providing the option to select different sub-genres within pop, or genres other than pop, could also be a useful feature. Of course, the feasibility of both of these features depends heavily on the size of the dataset, since most likely a separate model would need to be trained for each option (though a clever alternative could also be devised).

9 Conclusion

In this paper, I describe the process of iteratively designing a deep learning model for pop hook generation from scratch. The process involved constructing four models. I describe the merits and weaknesses of each and use these to inform design tweaks for subsequent models. I then create an original dance-pop song using a slightly modified version of a hook generated by my fourth and final model. Finally, I provide general insights gleaned through this process that may be of use to future researchers as well as suggestions for future work. While my final model still has limitations, its musical quality is vastly improved from my first model and is at a level where a songwriter and/or producer could potentially use it as a tool for kick-starting inspiration. Through the work presented in this paper, I hope to demonstrate the merits of deep learning for not only satisfying academic or technical curiosity, but for providing

tangible utility to artists as well.

References

[1] L. A. Hiller and L. M. Isaacson, *Experimental Music: Composition with an Electronic Computer*. McGraw Hill, 1959.

[2] C. Roads, *The Computer Music Tutorial*. MIT press, 1996.

[3] J. Briot, G. Hadjeres, and F. Pachet, "Deep learning techniques for music generation - A survey," *CoRR*, vol. abs/1709.01620, 2017.

[4] "Magenta: Make music and art using machine learning." Accessed on April 9, 2022. URL: https://magenta.tensorflow.org/.

[5] "Research areas: Machine learning." Accessed on April 9, 2022. URL: https://research.atspotify.com/machine-learning/.

[6] C. De Boom, S. Van Laere, T. Verbelen, and B. Dhoedt, "Rhythm, chord and melody generation for lead sheets using recurrent neural networks," *Machine Learning and Knowledge Discovery in Databases*, pp. 454–461, 2020.

[7] P. Roy, A. Papadopoulos, and F. Pachet, "Sampling variations of lead sheets," *arXiv preprint arXiv:1703.00760*, 2017.

[8] Y.-C. Yeh, W.-Y. Hsiao, S. Fukayama, T. Kitahara, B. Genchel, H.-M. Liu, H.-W. Dong, Y. Chen, T. Leong, and Y.-H. Yang, "Automatic melody harmonization with triad chords: A comparative study," *Journal of New Music Research*, vol. 50, no. 1, pp. 37–51, 2021.

[9] S. Oore, I. Simon, S. Dieleman, D. Eck, and K. Simonyan, "This time with feeling: Learning expressive musical performance," *Neural Computing and Applications*, vol. 32, no. 4, pp. 955–967, 2020.

[10] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[11] S. Ji, J. Luo, and X. Yang, "A comprehensive survey on deep music generation: Multi-level representations, algorithms, evaluations, and future directions," *CoRR*, vol. abs/2011.06801, 2020.

[12] J. Fossi, A. Dzwonkowski, and S. Othman, "Analyzing music genre popularity," in *Intelligent Computing*, pp. 284–294, Springer, 2021.

[13] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer: generating music with long-term structure," *arXiv preprint arXiv:1809.04281*, 2018.

[14] "Tabs that show the theory behind songs." Accessed March 19, 2022. URL: https://www.hooktheory.com/theorytab.

[15] "Hot 100 songs." Accessed March 19, 2022. URL: https://www.billboard.com/charts/year-end/2021/hot-100-songs/.

[16] G. Hadjeres, *Interactive deep generative models for symbolic music*. PhD thesis, Sorbonne Université, 2018.

[17] C. Raffel, *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. Columbia University, 2016.

[18] "Keras: the python deep learning api." Accessed on March 20, 2022. URL: https://keras.io/.

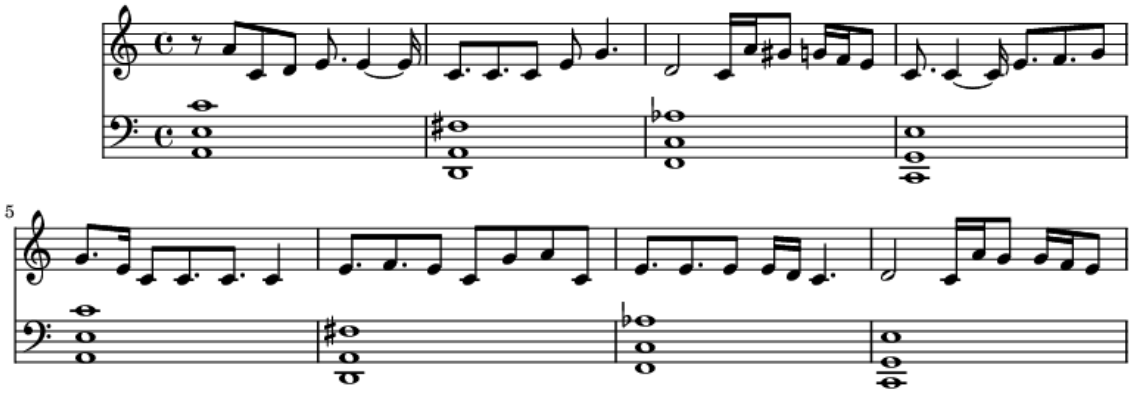
[19] D. Müllensiefen and M. Pendzich, "Court decisions on music plagiarism and the predictive value of similarity algorithms," *Musicae Scientiae*, vol. 13, no. 1_suppl, pp. 257–295, 2009.

[20] D. Castelvecchi, "Can we open the black box of ai?," *Nature News*, vol. 538, no. 7623, p. 20, 2016.

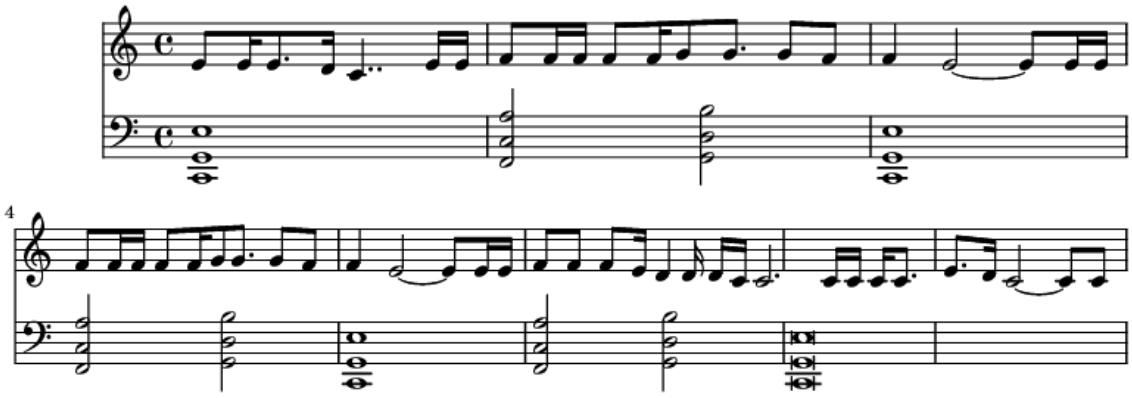
A Appendix

A.1 Plagiarism Check Results

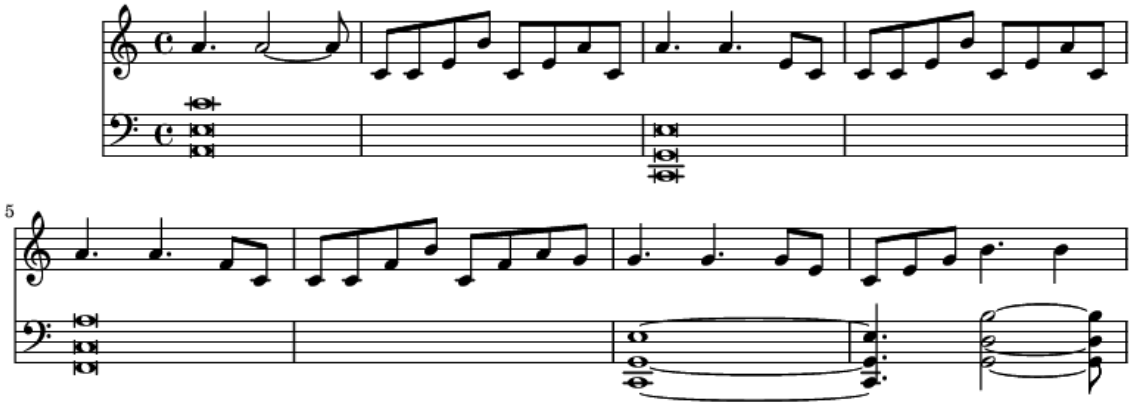
A.1.1 Top Three Similar Hooks to Figure 16b



"Say You'll Be There" by the Spice Girls. Similarity score: 57.6%

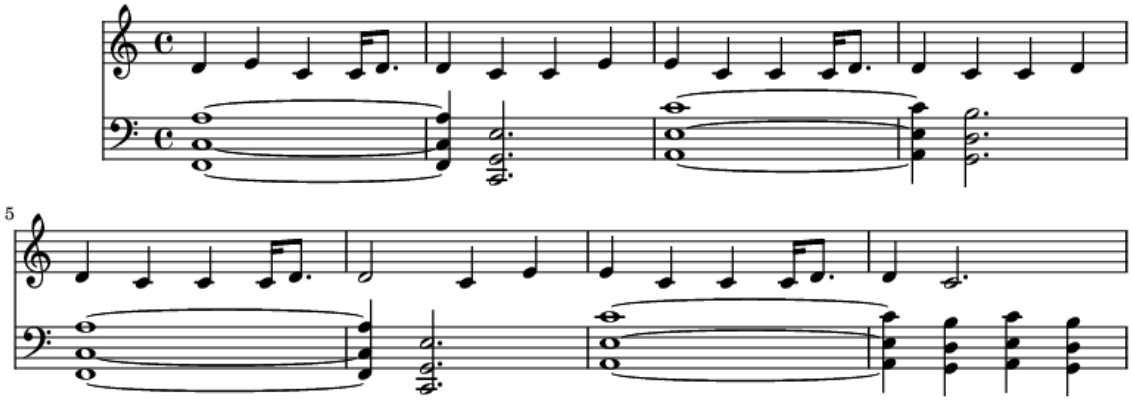


"Forever And For Always" by Shania Twain. Similarity score: 57.1%

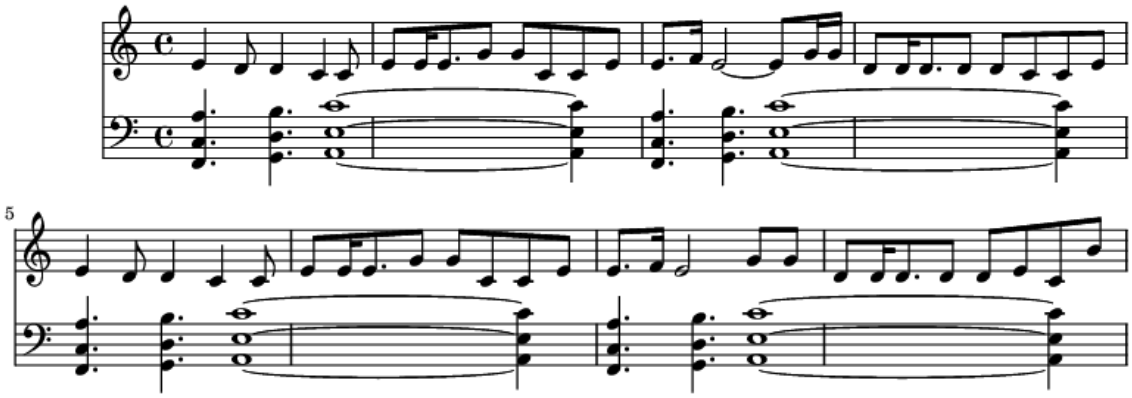


“1973” by James Blunt. Similarity score: 55.7%

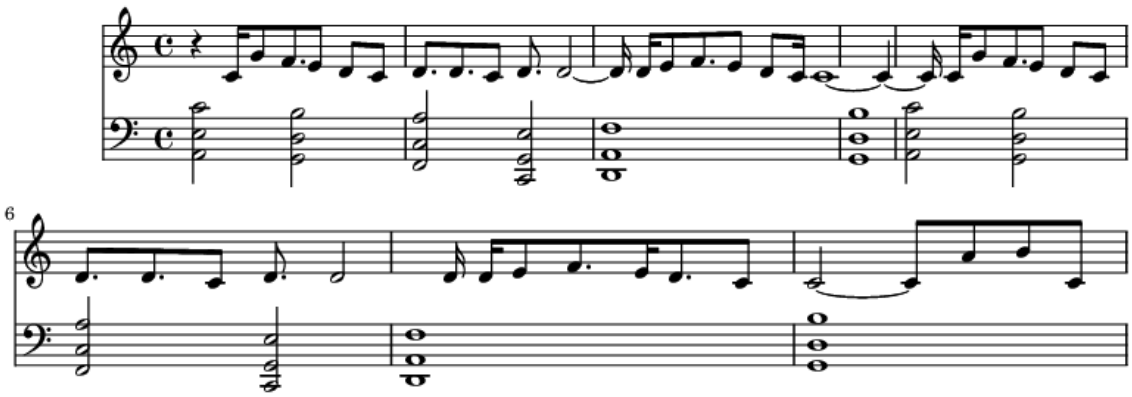
A.1.2 Top Three Similar Hooks to Figure 17b



"Closer" by The Chainsmokers. Similarity score: 70.0%

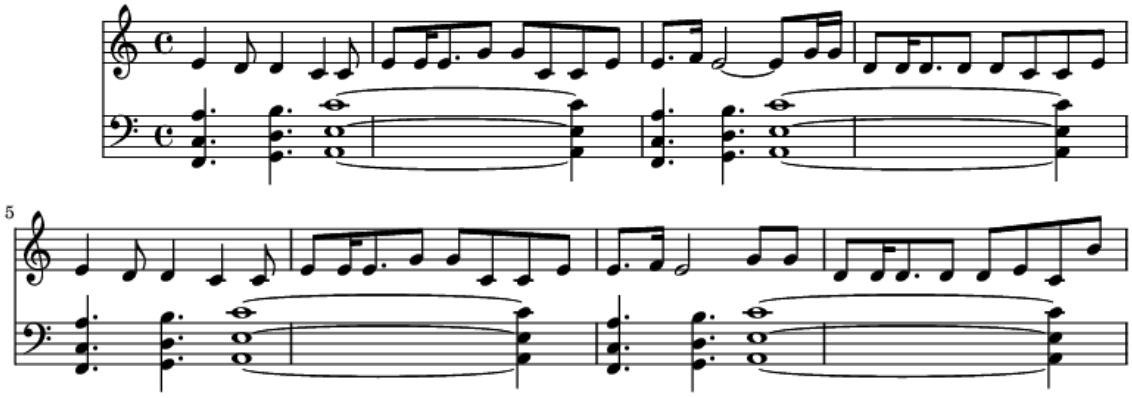


"Maps" by Maroon 5. Similarity score: 65.3%

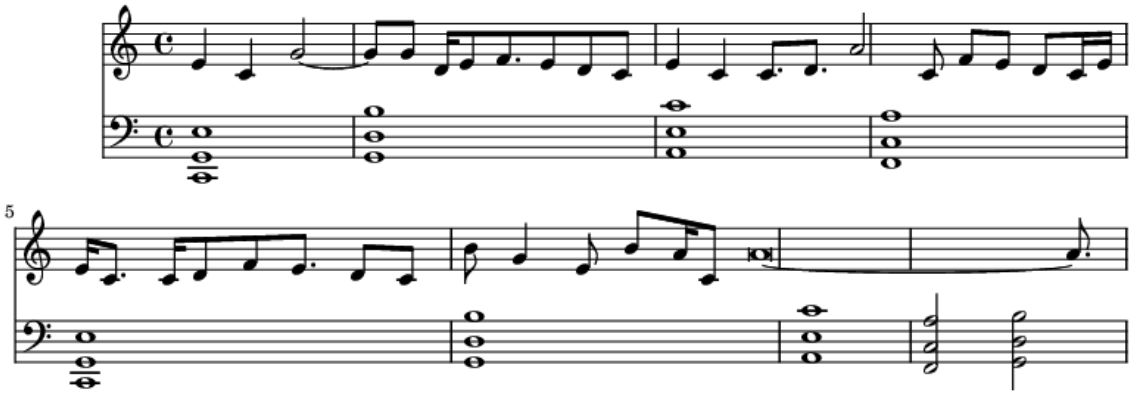


"Shelter" by Madeon and Porter Robinson. Similarity score: 64.2%

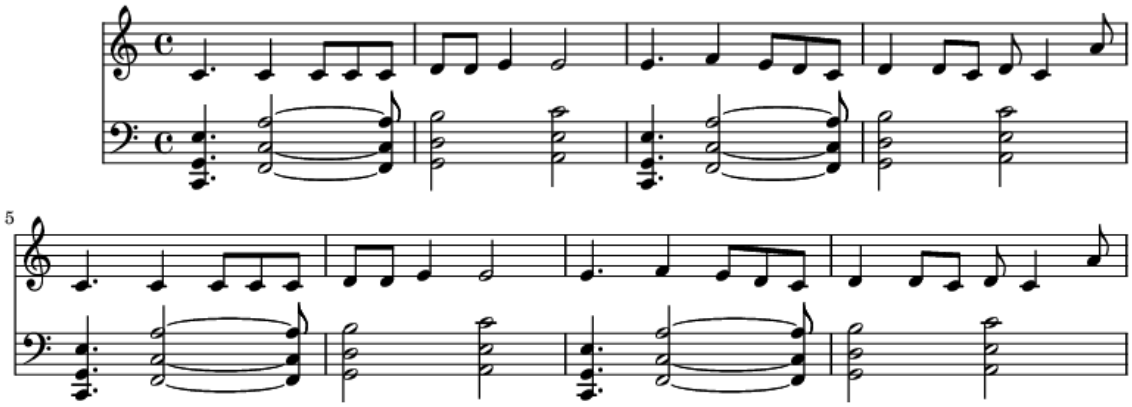
A.1.3 Top Three Similar Hooks to "Will You Learn" Hook



"Maps" by Maroon 5. Similarity score: 70.4%



"Hey Soul Sister" by Train. Similarity score: 70.0%



"Heaven Is A Place On Earth" by Belinda Carlisle. Similarity score: 69.2%

A.2 Source Code

All source code used in this paper, including executable Jupyter notebooks for all four models, can be accessed at the following link:
https://drive.google.com/drive/folders/1HjzQiuiBq9B8DKzrirJBAB0GPfb7Tnxg

A.3 Song Example Link

The song example featured in this paper, "Will You Learn", can be accessed at the following link:

https://soundcloud.com/dshen28/will-you-learn