The University of Edinburgh

School of Informatics

Master of Science

in

Artificial Intelligence

A.I. Enhancer — Harmonizing Melodies of Popular Songs with Sequence-to-Sequence

Author

Veselin Cholakov

August 2018

Abstract

The current dissertation project shows an exciting, never-before constructed implementation which writes and synchronizes appropriate, varied and imaginative *accompaniment* to musical melodies across genres. The melody can be polyphonic, of varied length, varied difficulty, in different key and time signature. The architecture of choice, a recurrent sequence-to-sequence model, is borrowed from the field of machine translation and has never before been used before in a musical context. The research has been made possible due to access to a unique dataset of 200 popular songs, each arranged for piano at a different level of performing difficulty (easy, intermediate, advanced), rich metadata and separate tracks for the melody and the accompaniment. A comprehensive software architecture, designed and implemented for the research, automated the collating, preprocessing, training and evaluation allowing work at scale. Lastly, besides merely implementing a novel system for melody harmonization, the current research produces close to a dozen conclusions which throw light on the strengths, limits, and modes of failure of recurrent neural networks in music.

Acknowledgements

I want to thank my supervisor Alan Smaill for trusting me and giving me freedom in this project, Matt Entwistle for taking me on the wild journey of a music dissertation, those who with small acts of kindness made the entire degree bearable, the group of friends who dragged me out of the library, and above all, my parents, and my sisters, for their immense love and patience.

Thank you.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Veselin Cholakov)

Contents

Part I
Introduction

The present research was conceived to give a directional answer whether the state-of-the-art machine techniques in translation can be re-purposed to arrange popular music by increasing the performing complexity of songs. In other words, the original question was whether given simple melodic cues, a recurrent neural network architecture can elaborate on the input by adding appropriate notes, harmonizing it with chords and creating a faster more difficult to perform rhythm. This is useful in educational settings where students can progressively practice with more difficult arrangements, as well as, as a time-saving tool for professional composers and arrangers.

The focus of the project shifted in the early proof-of-concept tests. The baseline model, a sequence-to-sequence architecture, showed remarkable results in writing an accompaniment to a given melody, without increasing the melody's complexity. Because the model tended to preserve the melody, the focus shifted to develop the best possible system for accompaniment – which like arrangement is another little-researched area with large practical potential.

The first half of the project was spent on devising the most suitable data representation to encode polyphonic music. This is a crucial step because sequence-to-sequence neural models are optimized to translate sentences of words, which always come one at a time – as opposed to notes, which can sound together at the same time. Just as importantly, 10 quantitative music-related metrics were created to track the performance of the current project.

The second half of the research project was spent on designing a robust and high-performance infrastructure. The research required software flexibility and scalable GPU-accelerated hardware to support the loading, queuing, configuring, training and evaluation of a vast number of experiments varying the data preprocessing specifications and tens of hyperparameters of five different model architectures.

The final solution, a Pyramidal RNN Encoder with Attentional RNN Decoder, surpassed the boldest initial expectations. The trained model has shown a remarkable ability to consistently provide musically interesting accompaniment to a wide range of melodies, some as simple as a handful of pitches, others as intricate as 20-second fast-paced advanced-level performances. The model faithfully reproduces the melody with an occasional improvisation and even a bit of a twist. The added chords are always sensible, even if occasionally a bit conservative. Finally, the results from more than 50

benchmarked experiments are analyzed and made available at the end of the current report shedding light how recurrent-neural network models process musical content in a symbolic form, which concepts they optimize first and where they still struggle.

Motivation

The last few years have seen a rising number of efforts to algorithmically compose music indistinguishable from the music created by humans. This is problematic on 3 counts, detailed in the next paragraph. The motivation of the current project is to refocus the research efforts in a different direction – on creating tools which empower creators of art, rather than imitating and replacing human creativity.

The problem with the current state of affairs is three-fold. First, we are not yet at the point where we can build a virtuoso AI which writes truly awe-inspiring compositions. Second, there is a dire need to first understand how neural models make sense of music on easier tasks before venturing after the holy grail. Third, even if engineers and developers find it fun to train neural networks to compose music in the style of the Beatles, it is worth asking how much the general audience actually wants that. The reason why fans follow bands on tours with zest and zeal arguably has as much to do with music as the lives of the musicians. How much would we care about a song that is recycled, rather than sourced from a daring place of great joy, wretchedness, despair, loss, passion, fear or love?

At the same time, there is many lower-hanging fruit which can be harvested. A worthwhile effort seems to be the pursuit of technologies which either automate repetitive tasks requiring relatively more limited amount of creativity (transcription, arrangement, orchestration), or models which work alongside with human creators (for example, by brainstorming motives or elaborating given musical ideas in notated or live form), or helping students learn in an educational setting. Such tools can help by saving teachers time and relieving music creators from run-of-the-mill jobs which hinder artistic creativity. Two recent (and rare) examples of such uses of technology in the field of music informatics are FlowComposer, a Markov-chain-based web applications which creates lead sheets, a form of music representation which keeps only the essential signatures of a track [Papadopoulos et al., 2016], and a model which automates the conversion of sheet music to guitar tablatures [Mistler, 2017], which is a preferred format by many guitarists [Barnes, 2014] [Obacom, 2015].

Scope

The overarching goal of the current research is the design and implementation of a working prototype of a model which allows a human and a computer to collaboratively create art. The objective of the current research is to ignite the creation of art through human-computer interaction via a *structural template*. The template can be as simple as few notes or as complex as an entire melody. A successful working model will recognize the provided template, elaborate on it and expand it by finding an appropriate chordal accompaniment, otherwise known as harmonization.

In order from minimum to ambitious, the objective can be broken down in three subgoals, in order from basic to hard:

1. Establish whether whether *seq-to-seq* models[1] can recognize a medium-long melody and reproduce a faithful rendition of it
2. *Sequence-to-sequence* is evaluated whether it can:
   (a) Enrich a melody tonally
   (b) Harmonize the melody with appropriate chords
   (c) Enrich the entire composition rhythmically
3. Finally, the study look into answering questions about the inner workings, the limits and the modes of failure of *sequence-to-sequence*, in particular:
   (a) What music concepts are easy for recurrent-neural-network-based models easiest to learn?
   (b) How does the model learn?
   (c) Whether metadata aids learning?

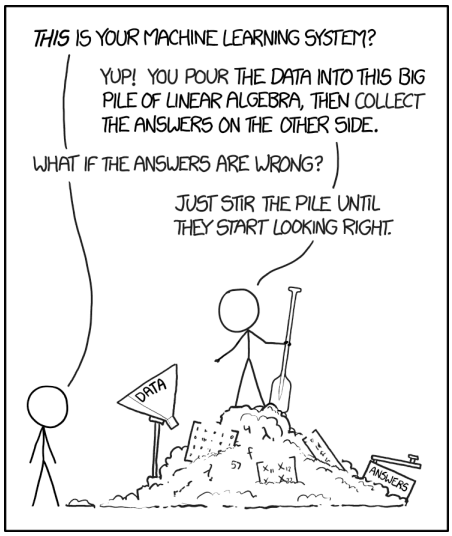


Figure 1: The ambition of the project at hand goes beyond than merely pouring data into a pot of linear algebra and stirring until the answers look right – a common criticism of the field of deep machine learning. One of the three principal objectives of the current study is to answer questions about the inner workings, the limits and the modes of failure of RNN-based models in the domain of computer-assisted music composition. Image source:XKCD.

[1] Described in detail in Part IV: Model

(d) What musical structures challenge the model?

(e) How aware is the model of different genres and how well does it re-conceptualize music in different genres?

Report Structure

The report is organized in eight chapters, including the current **Introduction**.

Problem Setup establishes important music concepts, the challenges of the field and presents previous related work.

Data is one most important parts. It presents the training dataset and its conversion to a suitable format for machine learning.

Model breaks down *Sequence-to-sequence* (also called *Encoder-Decoder*) and shows how all building blocks fit together in an end-to-end solution for encoding a sequence of any length and any type of data (words, letters, notes, pixels) and decoding that to another variable-length sequence of any data type.

Support Infrastructure details the implementation with strong focus on the pipelines for data preprocessing, the cloud-based infrastructure for training and comprehensive system for qualitative and quantitative evaluation.

Experiments lays out the queue of experiment which were ran and the evaluation metrics developed to gauge performance.

Results spells close to a dozen findings supported with data from experiments. Where possible illustrations and visualizations are provided to give the reader immediate intuition about the concepts.

Conclusion wraps everything together. The part begins with an analysis of the completed work and emphasis on the main take-aways. Limitations and speculation on next steps are given.

Part II
Problem Setup

1 Important Notions to Understand This Report

1.1 Music Theory Primer

Tonality (Key) Tonality is the organization of pitches and chords around a central note, the tonic. There are 12 distinct tonal groups and each identifies a set of pitches and chords which have audibly perceivable relationships between one another. All pitches and chords other than the tonic create a sense of tension which the tonic resolves bringing the piece to a point of rest [Schmeling, 2011]. The key of a composition is determined by the tonic note and can be in either *major* or *minor* mode, which is determined by an alteration of the third note of the scale. Generally, major compositions are brighter and celebratory, while minor compositions are perceived as darker and grimmer.

Meter (Time Signature) The meter of a piece, or its time signature, defines the rhythm of a piece. The time signature can be found at the beginning of the first staff as a fraction. The nominator determines the number of beats in a bar, and the denominator identifies which note duration gets one beat. For example, a time signature of 3/4 means that there are 3 beats per bar, and a quarter note gets one beat. A half note gets two beats, while an eight note gets half a beat. All bars contain the same number of beats [Handel, 1993].

Melody In popular music, the melody is a song's recognizable motif and it can be as simple as a few notes sounding one at a time. When someone is humming a song, they are most likely humming the melody.

Accompaniment The accompaniment plays a supportive role to the melody. Accompaniments vary across styles and historic time periods. In the music used for this project, the accompaniment enriches the audible quality of the melody both rhythmically and harmonically. The accompaniment provides the beat and outlines the chord progression, which either supports or contradicts the key.

Harmonization Harmonization is one of the components of accompaniment, specifically the chordal accompaniment. To create a sensible harmonization, a composer is

interested in the tonal center of the melody, its resolutions, the scale from which the melody derives its pitches, the possibilities allowed by the melody's rhythm, the style, and good judgment how many chords are too little and how many are too much.

Arrangement Arrangement is the process of reconceptualizing work for different instruments, levels of performing difficulty, or even genres. For example, the Jurassic Park theme song can be carried by an orchestra, or solo by a piano; if played on the piano, the soundtrack can be arranged to be easier to play, or it can sound richer which will increase its performing complexity. Additionally, a skilled composer can rewrite from one genre to another – taking Irish folk music and turning it into rock. All of those transformations are arrangements.

Transposition Transposition is the process of shifting a piece written in one key to another key. This is done with a straightforward mathematical operation which shifts all notes by a specified amount. This is analogous to changing a pitch's frequency in hertz by some multiplier.

In reality, all of the above definition are oversimplifications and each notion has produced disagreement and much scholarly debate. However, for the purposes of this dissertation, the above definitions provide a sufficient intuition about the workings of music.

1.2 Standard Music Representations

The purpose of this section is to draw a separation in the reader's mind that music is notes on a set of lines. Traditionally, music has been recorded in various systems. The invention of the computer did not change that. The main takeaway from this section is that there is not a single best representation. The question which we need to ask ourselves is what is our purpose, and what representation will serve our purpose best. A good representation is among the most import building blocks in machine learning, and Part III: Data looks carefully into crafting a representation suitable for encoding polyphonic music with recurrent neural networks. The current section glances over important traditional and digital representations of music which have informed the choice of the specific representation used in the current project.

Traditional

The Staff The staff is the most common system for recording Western Tonal Music. On a typical five-line staff, notes take different pitch based on which line or space they are positioned. While the vertical axis is the pitch, the horizontal axis denotes time. Non-local symbols carry information on the key, meter and tempo, as well as performance directions guiding dynamics, ornamentations, slurs and staccato, for example. But by and large, the system relies on explicit knowledge of the performer to bring a piece to life. The training data sourced for this project is originally available in a digitized staff notation.

Tablature More popular with guitarists today, this procedural representation instructs the player how to play (which fingers on what strings), as opposed to what to play (which pitches). Most curiously, rows of legendary guitarists cannot read traditional notated music – from the Fab Four of Liverpool and the King of Rock and Roll to Jimi Hendrix and Eric Clapton [Barnes, 2014] [Obacom, 2015].

In short, the staff notation is so ubiquitous and ingrained in Western societies that it is easy to forget that the staff is not the only way to transcribe music. Besides tablatures, other traditional representations include the Braille music for visually impaired musicians, various numbered systems like the Nashville number system, the lead sheet, figured bass, parsons code, percussion notation, Klavarskribo, graphic notation (such as Hans-Christoph Steiner's score *Solitude*, Rudolf Komorous's *Chanson*, and Iannis Xenakis' *Pithoprakta*).

Digital

MIDI MIDI is an industry standard protocol for communication between music software and hardware equipment. MIDI defines music as a stream of event messages in the form `<1920 On ch=1 n=69 v=67>`. The example signifies a *Note On* event at time 1920. A note identified by 69, (A4, 440 Hz), is played on channel 69 with velocity 67. MIDI specification allows encoding of additional data, which are outside the scope of this paper and are not covered here. The full MIDI specification is available online[2]. MIDI is an important format. First, it is a useful intermediary format when converting between systems. Second, the machine learning representation of the current project is inspired

[2]https://www.loc.gov/preservation/digital/formats/fdd/fdd000119.shtml

by the idea of MIDI's event messages. Details on the data representation used in the current project are available in 5.2. Training Representation.

Piano Roll Piano Roll is a representation medium which derives its name from the era of music boxes which played tunes with the help of perforated paper which encoded the pitches that were to be played. Piano roll is important for the current project because it provides useful means for visualizations and understanding how the trained model consumes and produces music. The x-axis on a piano roll represents time, while the vertical y-axis denotes the pitch. Each horizontal bar represents a note. The beginnings of bars correspond to *Note On* MIDI events, while the ends of bars correspond to a *Note Off* event. Piano roll is less intuitive to musicians than standard sheet notation, yet piano roll is more comprehensible than MIDI event messages. In other words, the format provides a good trade-off between readability and similarity to the data representation used to train the model. Piano roll is used extensible in the current report to visualize input and output.

Music XML Music XML is another standard format, derived from the more general specification of markup languages. MusicXML allows the transfer of data in a human-readable format which makes it popular for storage of notated musical composition. However, typically one is expected to convert MusicXML to a representation which is more suited to the task at hand.

2 Challenges

2.1 Music is Ambiguous

One the main problems in modeling music, just as trying to model any form of art, is its ambiguous and elusive nature. Think of a road trip when you listened to a song with your fellow passengers and everyone tapped at a different multiple of the same underlying beat. This happens because the *tactus*, the level at which a pulse is perceived, can be any level of the metrical hierarchy and different people respond to a different level. In other instances, metrical organizations are overlaid deceiving perception. When people cannot agree on the ground truth, it is even harder to teach machines to find it. The open interpretability of music makes it especially difficult to analyze because there are multiple defensible parsers and analyses of the same composition.

For all the progress in the past decade in voice recognition, to this date, transcribing music from audio to notated form remains an open research problem. In fact, two

musician listening to a song on the radio and transcribing it will make many judgments and will end up with slightly different scores. But even if a computer is spared the challenge of transcription and is only asked to identify the key of a given composition, the task is less than straightforward. Each key is identified by a set of notes, but both A minor and C major use the same set of notes. To distinguish between the two keys one must look at the *distribution* of pitches. Additionally, music writers use *accidentals*, non-matching notes which appear in the context of a key for artistic purposes. Modulation is another source of confusion. Modulations weaves a fabric and gives life to many musical compositions by changing the tonal center of a composition midway.

2.2 Representation is Hard

The previous section, Music Representation already took a deep dive into the various ways acoustic signals are represented symbolically. Part of the reason why there are so many ways is driven by the inability of any one representation to capture all of the variability. The loss of information is inevitable and some features, such as *timbre*, are practically impossible to capture. The inability to describe and define what makes a sound pleasurable creates a string of issues all the way downstream to evaluation.

Another problem stems from the fact that music typically requires two dimensions. Unlike natural language where words are always spoken one at a time, in music notes are played together more often than not. One can think of the standard sheet notation is having two axes – the horizontal axis of time and the vertical axis of pitch. Language, in comparison, is only defined by a time axis. Additionally, the language time axis is ordinal, such that only the oder of words matters, while in music precise timing is of paramount importance for rhythm.

2.3 Lack of Data

There are not large established datasets in the domain of music analogous to ImageNet in computer vision [Deng et al., 2009], the the Penn Treebank Project for Part Of Speech Tagging [Marcus et al., 1993] and MNIST for character recognition [LeCun and Cortes, 2010]. Music datasets are often built on an ad-hoc basis for specific projects. As a result, the datasets are highly specialized in one domain – folk tunes [Sturm et al., 2016], Mozart sonatas [Lattner et al., 2016] or Bach's chorales [Hadjeres and Pachet, 2016]. Consequently, research efforts are not concentrated and arguably this slows down progress. By way of comparison, until the 2009 introduction of ImageNet, object detection in images still relied on hand-engineered features and results were far from spectacular. However,

the introduction of the dataset ushered a number of competitions like the ImageNet Large Scale Visual Recognition Competition which attracted a large academic and corporate following. In just a few years, the object recognition accuracy jumped from 71% to 97% and even levels exceeding above human ability in narrow categories [Markoff, 2015].

2.4 No Commonly Accepted Benchmark

The challenge of benchmark is related to the challenge of missing data but it is also different. On one hand, the problem is the lack of clarity what should be measured, on the other hand is the issue of measuring. There is no good metric to measure creativity, or the agreeableness of a musical composition, or even the similarity between two melodies (though there are some attempts [Eerola et al., 2001]). While one could measure objective artifacts on tonal and rhythmic accuracy, they say little about the artistic qualities of a composition and how pleasurable it is to listen to. Music is still more an art than science and more about human perception than objective criteria. There has been use of various benchmarks in the academic literature – from KL divergence, which measures similarity of two probability distributions [Pachet and Aucouturier, 2004], to music Turing-style tests, the oldest one which seems to date as far back as 1964 [Ariza, 2009]. However, the fact is that neither of those have the status of gold standard, as is the BLEU score in machine translation [Papineni et al., 2002].

3 Related Work

The present research ventures into a novel area with little prior research. Generally all prior work falls loosely in one of two categories: composition of original music algorithmically from scratch, and music tools which do not require creativity (e.g. score following, transcription, chunking into meaningful units, key detection, beat detection, meter detection, etc.). The first group is most relevant for the current research, and thus the current section only looks into machine learning architectures which have shown ability to weave some sort of music structure which respects ideas in Western music theory.

3.1 The Big Picture

It must be noted that the idea of algorithmic, rule-based and probabilistic music composition has existed for centuries [Kirchmeyer, 1968]. The Ancient Greeks believed music and numbers are inseparably intertwined, Ptolemy and Plato theorized about rule-based composition, and Mozart practiced dice music, *Musikalisches Wurfelspiel*, combining mu-

sical fragments stochastically [Alpern, 1995]. As soon as computational resources were available, pioneering minds like Lejaren Hiller and Iannis Xenakis hopped on the train to exploit computational resources which aided generation of muscial textures statistically and structurally – for example, by stepping through paths of Markovian probability space and emitting sounds at each step [Gibson, 2011], or by varying the temperature in the Maxwell-Boltzmann equation of the distribution of the speed of molecules in gas [Antonopoulos, 2011, Luque, 2009]. Statistical efforts from the 50s were succeeded by shallow neural networks in the 1990s, and then once more the fashion changed at the turn of the 20th century when deep neural models became the song of the day. Even though modes have changed, it is important to remember that what was was hard years ago is still hard. Deep learning has only reduced or eliminated the need for hand-engineering features in some areas, yet none of the existing open research questions have been completely resolved. Yet, the new computational power has brought previously inaccessible areas within reach. The current research looks into some of those new frontiers while basing its efforts on prior work described below.

3.2 Melodic Harmonization

There a few early efforts which propose ways to find a chord progression given a melody. One of those originates from the University of Edinburgh [Allan and Williams, 2005]. Another, from the same year, proposes a standard Hidden Markov Model to label music in MIDI format with chord symbols [Raphael and Stoddard, 2004]. However, either of those attempts are relatively old and the task at hand is different. Annotating a MIDI track with chord symbols is easier than actually constructing the entire notated composition with the exact timing which is the goal of the current project.

That said, there are things to be glimpsed from those ideas. The proposed models observe pitches and produce labels which mathematically is a modeling of the transition probability distribution:

$$p(x_n|x_1, ...x_{n-1})$$

In other words, the emission probability distribution (the probability of predicting a given chord) is given by.

$$p(y_n|x_1, ..., x_n, y_1, ..., y_{n-1}) = p(y_n|x_n)$$

This is similar to the way recurrent neural networks model the probability to output an event at a given time steps. This has been one of the indications from the beginning

that RNNs are suitable for the tasks of the current project.

3.3 Algorithmic Composition

Most, if not all, prior research efforts in recent years attempt to generate original music algorithmically. Recurrent Neural Networks are the unreasonably effective backbone of any task which requires processing of any sequence [Karpathy, 2015] – from transcribing voice to text and decoding images to text for captioning, to predicting the next value in a sequence of time series data to know the price of a stock tomorrow. It does not come as a surprise then than RNNs dominate in attempts to model music.

RNNs are rarely used today in isolation. Instead, RNNs are re-purposed and affixed to a wealth of avant-garde architectures – in exotic combinations with generative adversarial networks [Mogren, 2016], autoencoders [Bretan et al., 2016] [Tikhonov and Yamshchikov, 2017], conditioning [Makris et al., 2017, Engel et al., 2017], sampling [Lattner et al., 2016] and even reinforcement learning [Jaques et al., 2016]. In all of the reviewed literature, with minor exceptions [Lattner et al., 2016, Yang et al., 2017], RNNs remain the workhorse, and any bells and whistles are performance boosters.

Eck et al's 2002 attempt was one of the earliest and best known demonstrations that Recurrent Neural Networks learn plausible transition probabilities between pitches on a training dataset in the genre of blues [Eck and Schmidhuber, 2002]. Later efforts established the flexibility of RNNs to learn basic music concepts such as chord and meter [Eck and Lapalme, 2008] and produce polyphony [Boulanger-Lewandowski et al., 2012]. The present project takes these findings and looks whether RNNs can learn without explicit guidance to change the meter of a song and add appropriate chords given a melody.

It is important to point out that all those prior efforts look at generating new music from scratch in the style of training corpus. Some have done better than others but the objective has been the same. Thus, in sourcing relevant ideas, the current research excavated bits and pieces of ideas buried in the main body of research publications which were headed in an entirely different direction. Of particular interests the author studied systems which allow for some level of *steering* the composition with human guidance. There are some remarkable attempts which allow steering of the generation with a user-defined key and meter [Lattner et al., 2016], musical motifs [Jaques et al., 2016], chord progression [Yang et al., 2017] or number of voices [Hadjeres and Pachet, 2016]. The work of the cited researchers is extremely exciting, and a tabular comparison between the different takes is made available the **Appendix** of the current report. However, much

more discussion will be spared on the current pages because the aim of the cited research is to generate new music, as opposed to *enhancing existing* musical compositions.

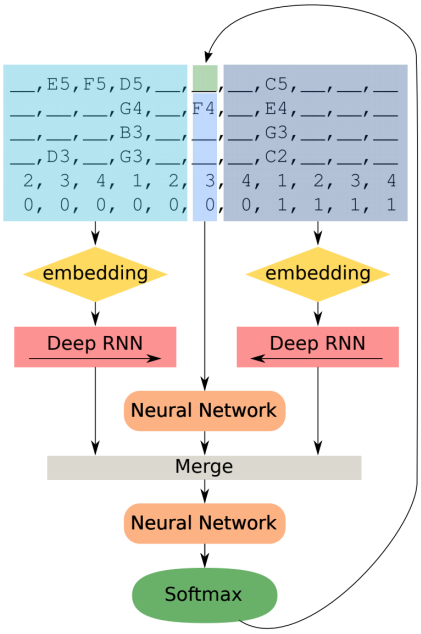


Figure 2: The architecture of DeepBach, a recurrent-neural network based model which can generate chorales in the style of Bach. Image source: [Hadjeres and Pachet, 2016]

The task the current project undertakes requires the trained model to show some level of understanding of the input and the *preservation* of its defining characteristics – from the key and the rhythm to the melodic motifs. In other words, the need to encode a representation of the input and then output a rendition of it, has naturally led the author to research one particular type of recurrent neural network architectures: sequence-to-sequence, discussed immediately in the next section.

3.4 Ideas from Machine Translation

The last few years have been turbulent for statistical machine translation with the emergence of example-based machine translation, mostly centered as of recently on variations of sequence-to-sequence architectures [Sutskever et al., 2014] [Kalchbrenner and Blunsom, 2013]. In the short span since 2014, new ideas have swiftly replaced traditional feature-engineered ad hoc translation system [Koehn et al., 2003]. In contrast to the old feature-engineered phrase-bank models, in which each component is tuned independently from the rest, neural sequence to sequence systems have offered to train a single neural network. Since Sutskever, Vinyals and Le's 2014 introduction of he sequence-to-sequence model (also called encoder-decoder), it has become the de-facto standard with implementations in the most popular machine learning toolkits like TensorFlow and PyTorch.

Even though, as shown in the previous section, recurrent neural networks are the backbone of all successful music generation models, they have never been used before in a sequence-to-sequence configuration in musical context. To the author's knowledge, the current research is the first attempt to *translate* between arrangements of the same song. The current project presents very exciting opportunity to measure the effectiveness of neural systems in preserving and recognizing the core ideas in the musical surface, while

altering the presentation.

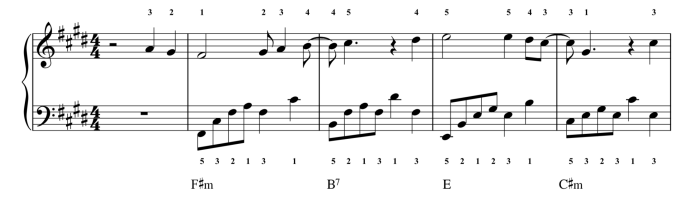
The only two known prior use of some kind of encoder-decoder architecture for music synthesis are the Variational Recurrent Auto-Encoder (VRAE) [Fabius and van Amersfoort, 2014], and the VRASH Melody Generation System [Tikhonov and Yamshchikov, 2017]. However, it must be emphasized that VRASH and VRAE use a different encoder-decoder model than the architecture proposed by Sutskever et al. which is backbone of machine translation today. Variational methods map the encoded data to a *distribution* over latent variables [Fabius and van Amersfoort, 2014].

Part III

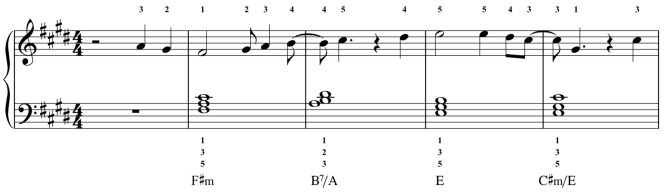
Data

4 Dataset Description

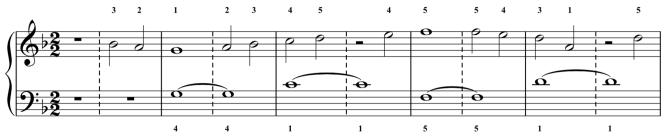
The current project had access to a never-before-used dataset of 200 commercial songs kindly provided by a New York-based educational company which offers online piano lessons. Professional composers, hired by the company, have arranged hundreds of popular songs across genres appealing to a wide range of demographics. The songs are arranged at different levels of performing difficulty – beginner, intermediate and advanced. In general, lower levels prefer slower notes and avoid chords, or use easier to play chords in terms of positioning of the fingers on the piano keyboard. The beginner level is the most different level as it does not use chords at all and the melody is monophonic with only one note sounding at a given time.



(a) Advanced Arrangement



(b) Intermediate Arrangement



(c) Easy Arrangement

Figure 3: A sample from the dataset, the first five bars of *All My Loving* by John Lennon and Paul McCartney in three levels of performing difficulty. This illustration has been previously published with the project's proposal report.

Figure 3 shows the first few measures of *All My Loving* by John Lennon and Paul McCartney in easy, intermediate and advanced arrangement. The audible quality of the easy composition is most distant to what most listeners would recognize listening to the original on the radio or on YouTube. That said, it is clear to someone musically competent that the same melodic motifs appear in all three arrangements of the provided sample. In fact, if that notated music were to be synthesized, most people without music education would intuitively hear the commonalities between the three arrangements.

Genre	No. Easy	No. Intermediate	No. Advanced
Classical	16	16	13
Contemporary Piano	6	6	6
Country	8	8	4
Film & TV	13	13	12
Musical Theater	12	12	9
Pop	52	52	31
Religious	5	5	5
Rock	57	57	22
Traditional & Holiday	15	15	4
TOTAL	**184**	**184**	**106**

Table 1: Breakdown of the songs in the dataset by genre and level of performing difficulty. Not all songs are available in all three arrangements but all songs are available in at least two. This illustration has been previously published with the project's proposal report.

5 Data Representation

5.1 Disk Storage Format

The dataset was originally provided as Sibelius (`.sib`) files, a proprietary file format of Avid Technology. The software is well-known among professional composers and it is the best-selling notation software. It allows composers to write music on a computer and the software takes care of the formatting, layout and typography. The file format, however, is completely unsuitable for machine learning. First, it is proprietary which makes the fies unreadable without expensive software. On the other hand, the files are extremely heavy and seem to make inefficient use of disk storage.

The first task in the project has been exporting all Sibelius files to MusicXML. This reduced the dataset from 10.4 Gigabytes to just over 1.3 Gigabytes, an almost ten-fold decrease in disk storage. MusicXML is a good intermediary format with wide support. However, it requires the development of custom parsing functions which is time-consuming and error-prone process.

Recent years have seen the open-sourcing of Google's protocol buffers, serialized data storage containers which are lightweight and neutral to platform and programming language. In Python, which has been the language for the implementation of the current project, protocol buffers function very similarly to native iterable data structures with named fields (e.g. dictionaries). Protocol Buffers are used extensively in Google internally, and externally in open-sourced projects.

Figure 4: Conversion to an efficient data-storage file format allowed the reduction of the footprint of the original dataset from 10.4 Gigabytes to less than 46 Megabytes, a 226 times decrease.

The Magenta project[3], Google's stab at creating art and music with machine learning, has open-sourced useful methods to easily convert MusicXML and MIDI into *NoteSequences*, an implementation in the serialized protocol buffers format with pre-defined fields for notes of a given pith, start time, end time, time signature, as well as beats per minute, key signature and tempo. The conversion from XML to NoteSeq is very fast. The conversion of the original dataset from MusicXML to NoteSequence reduced the total disk use from 1.3 Gigabytes to less than 50 Metabytes, a 28-times decrease. This is due to the fact that a lot of unnecessary XML data and all of the redundant tags are discarded. Figure 4 shows the three data formats with short snippets underneath illustrating the way data is stored, displayed and used.

[3]https://magenta.tensorflow.org/

5.2 Training Representation

Intuitive Explanation

The conversion of the dataset from Sibelius to MusicXML to NoteSequence increased the speed and ease of loading, collating and preprocessing data. However, NoteSequence is still just a way to store data on a disk, and it cannot be inputted directly into a model for training.

The model used in the current research, a sequence-to-sequence architecture, detailed in section 6. Model Definition, expects to receive input in the form of a sequence of events. In other words, the model can read only one event a time, one after another and the model does not have an intrinsic notion of clock time. This creates two problems. First, while the model understands the order of events, it cannot represent metrical time which is crucial for music. Without a notion of time, rhythm is completely monotonic and all notes have the same duration. The second problem is that of polyphony. Sequence-to-sequence can receive only one event at a time. So, how do we encode two notes played at the same time?

Both problems can be solved by taking inspiration from the MIDI representation introduced in 2.2 Standard Music Representations. We can use an event-based representation with three types of events: `ON`, `OFF` and `<SHIFT>`. Each event type represents a key that is associated with a value:

```
<ON, Pitch [1-127]>
<OFF, Pitch [1-127]>
<SHIFT, Value [1-16]>
```

`<ON 60>` means start playing the pitch identified by the value of 60 (middle C). `<OFF 60>` means stop playing middle C. To play C major chord C-E-G, we can encode a sequence of three events `<ON 60>, <ON 64>, <ON 67>`. Notice that the three events are inputted into the model one after another. However, we can agree to accept that all `<ON>` and `<OFF>` events happen at a single time step, unless a `<SHIFT>` event type separates them.

Each encoded sequence always begins at time 0 and `<SHIFT>` moves the clock hands by some interval specified by the value that is attached to the `<SHIFT>` event (i.e. `<SHIFT 16>` or `<SHIFT 8>`). A few shifting events one after another of different value are completely acceptable and rather common. It is important to underscore that `<SHIFT>` does not use clock time measured in seconds, but rather metrical time defined by the denominator of the time signatures. `<SHIFT 16>` means advance one time step of the duration

of a sixteenth note (semiquaver). This is the smallest subdivision supported by the implementation. It has been absolutely sufficient because the semiquaver is the shortest note that appears in the training dataset.

Implementation

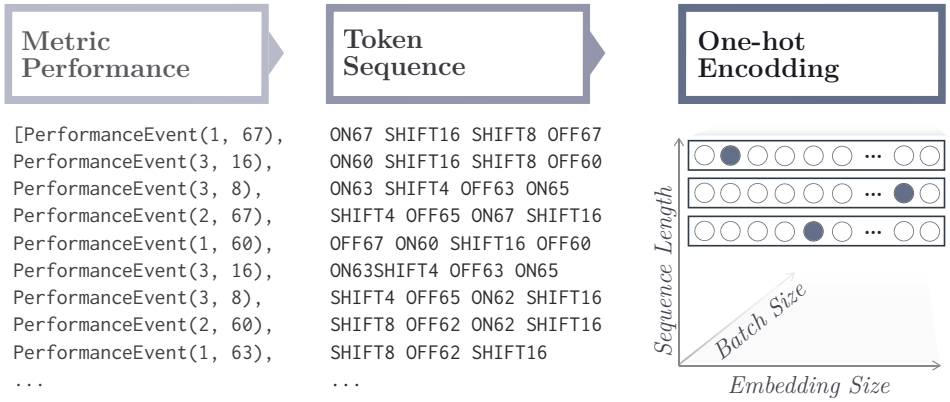


Figure 5: An illustration showing three data representations used in the course of the project. All representations encode the same information but are used for different purposes. Metric Performance is used during preprocessing. Token Sequence is an intermediary format for storing on disk and debugging. One-hot Encoding is the final representation which is fed to the trained model.

Now that we have developed the intuition on the chosen training representation, we look how the representation is implemented and used internally.

Figure 5 shows (in order) an example of a Metric Performance, Token Sequence and One-hot Encoding. All representations carry the same information. The only difference is how they are used under the hod. NoteSequences are converted to a Metric Performances, a Python class which stores a track of music as a list of events. Many preprocessing manipulations are done on the Metric Performance class. Token Sequence is the export format which converts individual events to space-separated tokens and saved as a text file on disk. Token Sequence has been defined to most closely resemble the standard format for data representation in natural language processing tasks.

We can think of each token (i.e. `ON67`) as a word. In general, the vocabulary of music events is much smaller than the vocabulary in natural languages. Only about 272 distinct events are recorded at most (128 "words" for turning notes on, another 128 tokens for turning notes of, and 16 events for shifting metrical time). In practice, the vocabulary file is even smaller because a far smaller range of pitches is used (a piano has 88 keys, not 128).

Additional metadata tokens may be added to the vocabulary of "words". This was done in the course of the current project. Metadata tokens were used to condition the encoding and the decoding of the entire sequence on, say, a genre (i.e. pop, rock, religious), or a time signature. The details are laid out in the Core Tests section in Part VI: Experiments.

Finally, the last column in Fig. 5 shows that the data representation which was fed to the trained model. Each token from the vocabulary is associated uniquely with a one-hot encoded vector [Harris and Harris, 2010]. Vectors are arranged in a 3-dimensional matrix. The matrix's horizontal dimension is the **Embedding Size**, the vertical dimension is the **Sequence Length**, and the third dimension is the **Batch Size**.

Embedding Size equals the number of "words" in the vocabulary file (in our case, about 270). Each word in the vocabulary is converted to a one-hot-encoded vector. The length of each vector is the length the embeddings size. Zeros are inserted in all position, except the position which identifies the token. A lookup function allows conversion between a one-hot encoded representation and the words in the vocabulary. Once a model is trained with a given vocabulary, inference can only be performed using the same dictionary.

Sequence Length is determined by the length of sequence of tokens. For each "word" in the vocabulary, an embedding vector is inserted from top to bottom. Because sequences are of different length,s and a matrix is of a fixed size, the length of the longest sequence is taken, and shorter sequences are padded. The Sequence Length Dimension typically varies across batches and equals the length of the longest sequence.

Batch Size allows several songs to fed to the model at the same time. The loss and gradient values are calculated on the entire batch. The batch size is a user-configurable hyper-parameter, generally a power of two and in the range between 8 and 512. Higher values lead to faster convergence but consume more memory. The batch size can be changed during training. Batching together many examples in the beginning allows faster convergence. Once the loss decreases, the batch size can be reduces to allows more fine-grained search of the local minima.

Part IV
Model

6 Model Definition

6.1 Overview

This chapter presents the building blocks of an *Encoder-Decoder* architecture, also known as *Sequence-to-Sequence* or *seq2seq*.

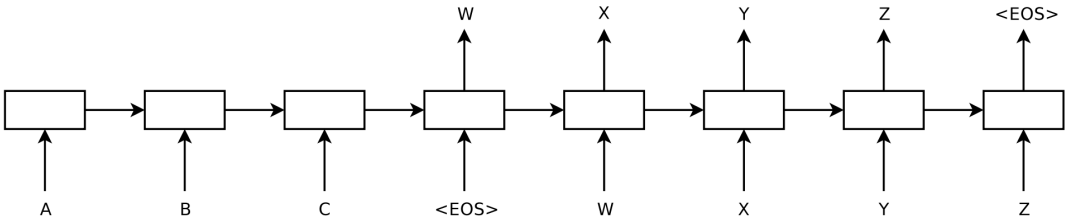


Figure 6: The original illustration which introduced sequence-to-sequence architectures [Sutskever et al., 2014]. A sequence of input events A, B and C is consumed one by one and encoded into a *thought vector*. A special End-Of-String token, `<EOS>`, indicates the end of the encoding process and the start of the decoding process which produces W, X, Y, and Z. The biggest benefit of the architecture is that it allows variable-length input to be transformed to variable-length output.

The success of the framework can be attributed to its generality and the minimum assumption made on the type of sequence that is encoded. The architecture has been successfully used before in domains as varied as machine translation [Luong et al., 2014] [Jean et al., 2014], captioning [Vinyals et al., 2015b, Xu et al., 2015], parsing [Vinyals et al., 2015a] and conversational modeling [Vinyals and Le, 2015].

The essence of the idea is to use a nonlinear function, typically a multi-layered neural network, to encode an input sequence into a vector of fixed length, which can be used to initialize the first hidden state of a second neural network which unrolls the encoded representation into the target sequence (see Fig. 6).

Mathematically, the objective is to maximize the conditional probability:

$$p(y_1, ..., y_{|Y|}|x_1, ..., x_{|X|}) = \prod_{t=1}^{|Y|} p(y_t|x, y_{t-1}, \ldots, y_1) \tag{1}$$

where $x_1, ..., x_{|X|}$ is the input sequence, and $x_1, ..., y_{|Y|}$ is the target sequence. Notice how each element of the target sequence is conditioned on both the entire input sequence

and all of the elements in the target sequence preceding the the element which we are predicting at the current time step.

In practice, we rewrite Equation (1) as a log probability:

$$\log p(y_1, ..., y_{|Y|}|x_1, ..., x_{|X|}) = \sum_{t=1}^{|Y|} \log p(y_t|x, y_{t-1}, \ldots, y_1) \tag{2}$$

In other words, our goal is to find:

$$-\frac{1}{|X|} \sum_{Y,X} \log p(Y|X)$$

6.2 Components

Softmax

Throughout this chapter, the softmax function is often used to normalize the output from the neural network into a vector of probabilities which add up to 1. The softmax is defined as:

$$softmax(y) = \frac{\exp y}{\sum_{y' \in Y} \exp y'}, \tag{3}$$

The softmax function is useful to normalize the output from our neural networks to a vector of probabilities. Each position in the vector indicates the likelihood of an event. Selecting the event with the highest likelihood with an $argmax$ function allows us to predict the most probable event.

Recurrent Multi-layered Function

Most commonly, both the encoder and decoder of sequence-to-sequence models are recurrent neural networks (RNNs), which are a generalization of a feedforward neural network with multiple time steps. Given an input sequence $x_1, ...x_t..., x_{|X|}$, an RNN calculates the hidden states h_t and the output y_t as follows:

$$h_t = act(W^{hx} x_t + W^{hh} h_{t-1} + b_h) \tag{4}$$

$$y_t = W^{yh} h_t, \tag{5}$$

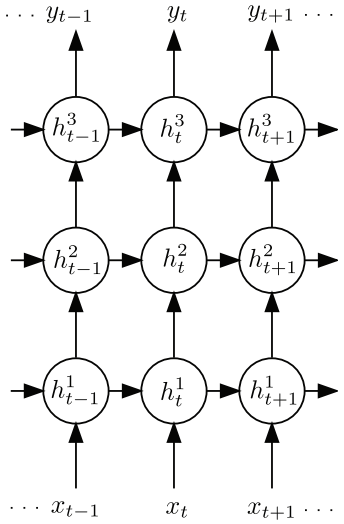


Figure 7: Picture of an unfolded deep RNN. The network has three hidden layers (h^1, h^2, h^3). It receives consecutively one-hot encoded vectors x_{t-1}, x_t, x_{t+1} and outputs y_{t-1}, y_t, y_{t+1}. Diagram source [Graves et al., 2013a].

where b_h is a bias term, W are learnt weight matrices, and *act* is some activation function, most commonly element-wise application of the sigmoid function (see Fig. 7) [Graves et al., 2013b].

Neural Computational Units (LSTM)

In practice, plain RNNs suffer from the vanishing gradient problem which makes impossible the learning of long-term dependencies [Pascanu et al., 2013]. For this reason, rather than the simple version of the hidden states defined earlier, we change the definition to use a more complex type of a computational unit: Long Short Term Memory (LSTM) cells which control the flow of information via an *input*, an *output* and a *forget* gate (see Fig. 8) [Hochreiter and Schmidhuber, 1997]. LSTM neurons are implemented with the composite function below [Graves et al., 2013b]:

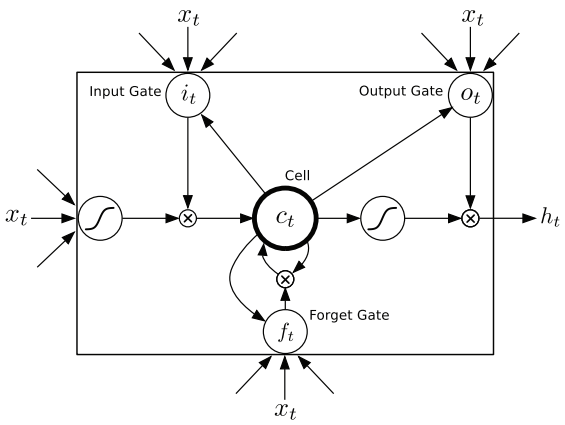


Figure 8: Diagram of a Long Short Term Memory neuron. Source: [Graves et al., 2013b].

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{6}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{7}$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{8}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{9}$$

$$h_t = o_t \tanh(c_t) \tag{10}$$

Encoder

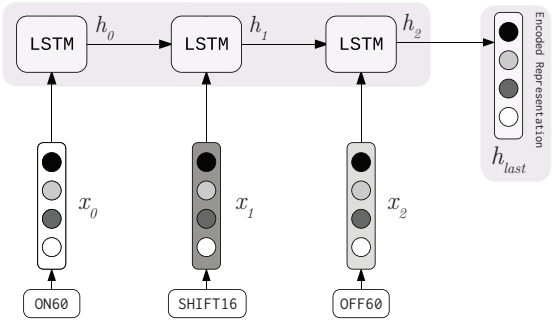


Figure 9: A diagram of an RNN LSTM encoder which is consuming sequence of three music events. The picture shows only a single row of LSTM cells for simplification of the diagram. In practice, we will use no less than 2 and we experiment with up to 4 layers in the current project. Original Illustration by Guillaume Genthial: [Genthial, 2017]

Now that we have defined LSTM RNNs, we are ready to encode a toy example with an LSTM RNN. The three tokens `ON60, SHIFT16` and `OFF60` encode the playing of middle C for a duration of a 1/16 metrical time. First, the pitch identified by a value of 60 is turned on. Then, a duration of length 1/16 metrical times elapses [`SHIFT16`]. Following that, the note at pitch 60 is released [`OFF60`]. Each of the three tokens is associated with a d-dimensional one-hot encoded vector $x_t \in \mathbb{R}^d$ via a lookup table. Each vector is fed into an LSTM RNN one at a time and the final hidden state h_{last} is the aforementioned *thought vector* which encapsulates the meaning of the entire input sequence of three events (see Fig. 9). The encoding stops once a special token `<EOS>` is encountered. Once the encoding stops, we are ready to take the final thought vector to our Decoder.

Decoder

We take the last hidden state from of Encoder, h_{last}, which holds a compacted representation of the entire input sequence. We initialize the hidden state of the first neuron of

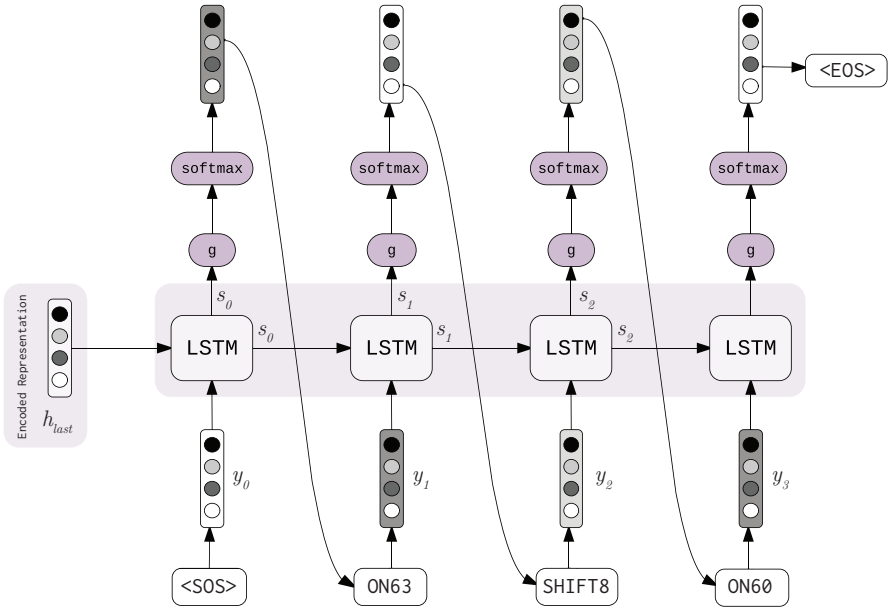


Figure 10: A diagram of an RNN LSTM decoder which is unrolling the thought vector into a sequence of four events. The output from each step becomes the input to the next decoding step. Illustration by Guillaume Genthial: [Genthial, 2017]

the Decoder with h_{last}. At the same time, we feed into the first neuron of the Decoder a specially designated character `<SOS>` which indicates starts of the decoding process. Mathematically, the Decoder is initialized as follows:

$$h_0 = LSTM(e, w_{<SOS>}) \tag{11}$$

$$s_0 = g(h_0) \tag{12}$$

$$p_0 = softmax(s_0) \tag{13}$$

$$i_0 = argmax(p_0), \tag{14}$$

where g is some affinity function which reshapes h_0 to a vector of the same size as the token embeddings which is a user-configurable hyperparameter (d-dimensional vectors). In subsequent time-steps, the variables are updated as follows:

$$h_t = LSTM(h_{t-1}, w_{i_{t-1}}) \tag{15}$$

$$s_t = g(h_t) \tag{16}$$

$$p_t = softmax(s_t) \tag{17}$$

$$i_t = argmax(p_t) \tag{18}$$

6.3 Training

The model is trained to maximize the probability of the target sequence, or equivalently to minimize the cross entropy between the predicted and the target distribution:

$$-\sum_{t=1}^{|Y|} \log p_t[y_t] \tag{19}$$

Traditionally, the optimization, which updates the network weight based on training data, has been done with Stochastic Gradient Descent. In recent years, the procedure has been superseded by the Adam optimizer which boosts adaptive learning rate for each parameter and behaves likes a heavy ball rolling downhill with a momentum determined by an exponentially decaying average of the previous squared gradients [Kingma and Ba, 2014]. Adam is preferred due to its computational and memory efficiency and ability to optimize well in a wide range of circumstances [Brownlee, 2017].

7 Enhancements

7.1 Attentional Decoder

Attention is a vector which is calculated from the output of a softmax from a dense layer of the sequence-to-sequence model. Attention mitigates the issue of the vanishing gradient by allowing the trained model to focus more easily its on different parts of a long input sequence [Bahdanau et al., 2014].

For each target that is decoded, the attention vector, also called a *context vector*, calculates a probability distribution by taking the input of all encoding cells. This gives the model more of an overview of the input in addition the left-right encoding captured in the thought vector. Mathematically, the context vector can be calculated as follows:

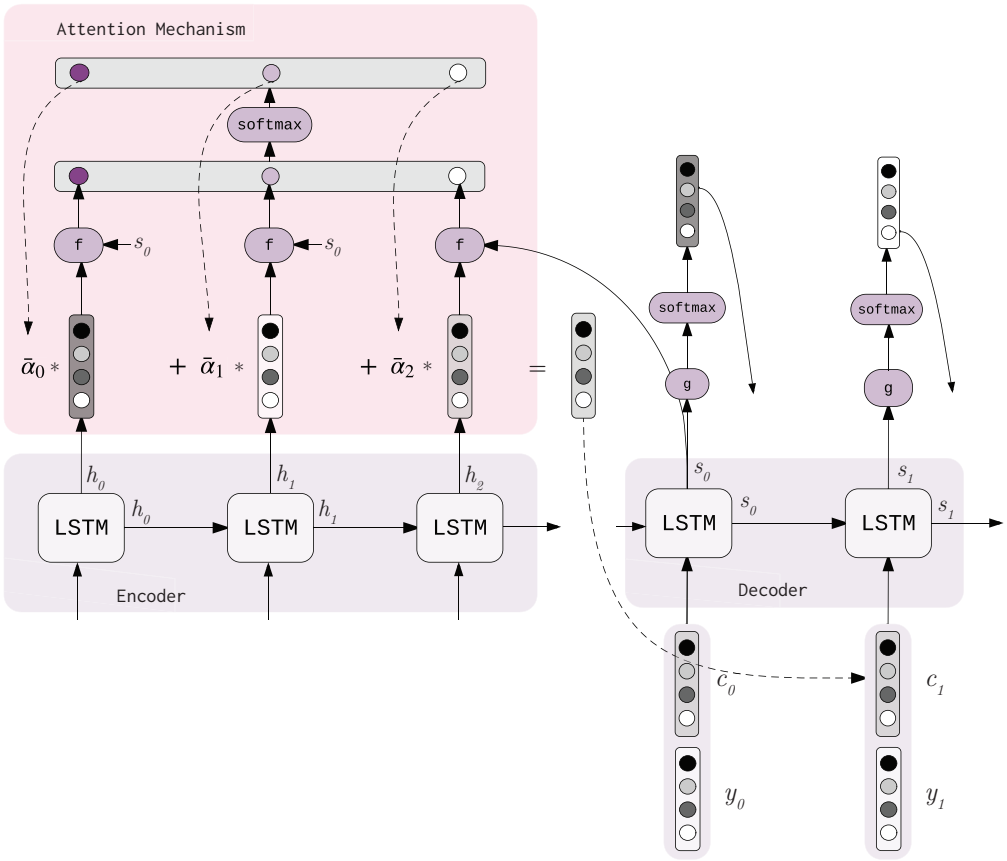


Figure 11: A diagram showing the attention mechanism. The picture has been expanded and customized based on an original illustration by [Genthial, 2017]

$$a_t = f([h_{t-1}, e_t] \tag{20}$$

$$\bar{a_t} = \text{softmax}(a_t) \tag{21}$$

$$c_t = \sum_{t=0}^{n} \bar{a_t} e_t \tag{22}$$

The function f is typically a dot product or concatenating function. The output from f is passed through a normalizing softmax layer and the result from the softmax is used to calculate a weighted average of the input encodings. Then, the context vector c_t is fed to the decoding LSTM. To be able to feed c_t to the Decoder, we alter slightly Eq. 15 to accept the context vector as an input [Bahdanau et al., 2014].

$$h_t = LSTM(h_{t-1}, [w_{i_{t-1}}, \mathbf{c_t}]) \tag{23}$$

An interesting by-product of attention is the ability to visualize the spread of attention to get an insight what the model is learning. In fact, the current report concludes with an attention visualizations which shows how the trained model distributes its attention when deciding what chords to harmonize a melody with 13 Inside the Black Box .

7.2 Bidirectional Encoder

Bidirectional RNNs (BRNNs) were first conceived to expose the model to additional data from both past and future states [Schuster and Paliwal, 1997]. This is achieved by constructing two hidden layers in opposite directions.

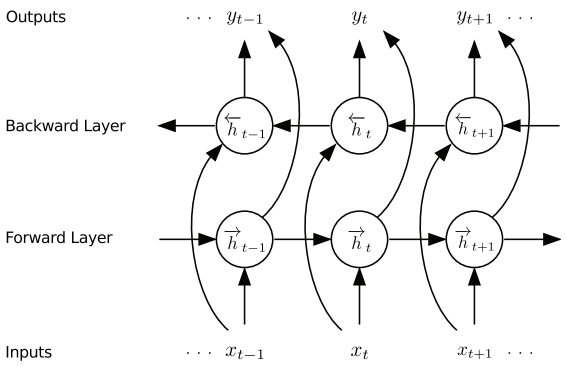


Figure 12: Bidirectional RNNs allow a model to take information from earlier and later points in the sequence. Diagram Source: [Graves et al., 2013a]

BRNNs have been shown to boost performance for tasks where context is important, such as speech, handwriting, named entities recognition, as well as predicting the structure of protein [Graves et al., 2013b]. Context matters tremendously in music and bi-directionality should be beneficial. The full set of equations for implementing a Bidirectional RNN Decoder is available in an easy-to-understand way by [Graves et al., 2013a].

7.3 Pyramidal RNN Encoder

The Pyramidal RNN architecture, which is used as an Encoder, is less well-known that the models defined above. However, specifically in the context of modeling notated music, there are reasons to believe that the architecture can boost performance because of the hierarchical nature of music. The background which has informed the speculation rests with the widely influential *Generative Theory of Tonal Music* [Lerdahl and Jackendoff, 1985].

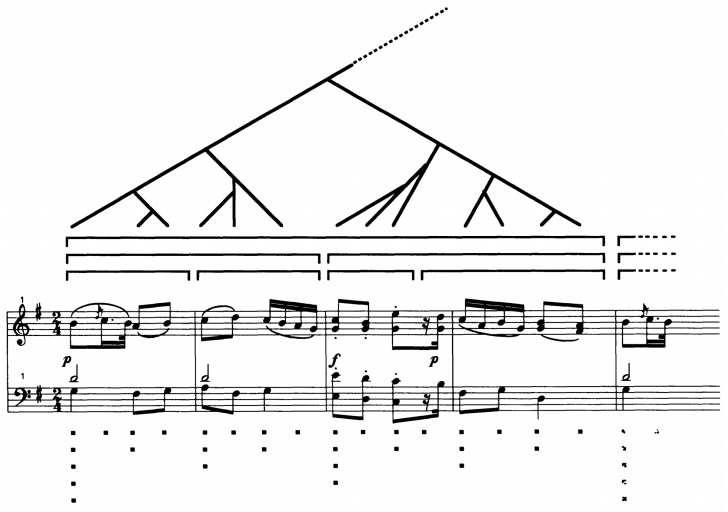


Figure 13: The illustration shows an applied example of Lerdahl and Jackendoff's Generative Theory of Tonal Music. Time-Span Reduction is the the tree-like structure on top, Grouping is the the square brackets underneath the tree, and the Metrical structures are represented as vertical dots underneath the staff. [Fred Lerdahl, 1983]. Image source: [Cross, 1998]

Lerdahl and Jackendoff's Generative Theory of Tonal Music

Lerdahl and Jackendoff's Generative Theory of Tonal Music has laid out a comprehensive attempt to relate a listener's intuitive music comprehension to rules and processes subject to formal analysis. It comes from well before the time of deep learning and to the author's knowledge it has never been tested in that context. The current project is presents a good opportunity to take a classical theory and combine it with a recurrent architecture which incorporates some of Lerdahl and Jackendoff's ideas.

In short, the theory is anchored in the notions of the Grouping, Metrical, Time-Span and Prolongational structures. The Grouping Structure aims to segment the musical surface into sections, phrases and motives based on rules about proximity, change, intensification, symmetry and parallelism of pitch events. The Metrical Structure organizes music into weak and strong beats dealing with a temporal framework (rather than pitch). Time-Span and Prolongation are higher order abstractions, predicated on the Metrical & Grouping organizations. Time-Span Reduction addresses rhythm by looking into pitch and harmony and defining parts into rhythmic hierarchy. Prolongation additionally builds on top of Time-Span reduction to capture increasing and decreasing tension, thus encoding a sense of music flow across phrases.

Hierarchical RNN

Hierarchical RNNs resemble Lerdahl and Jackendoff's Time-Span reduction structure and are evaluated in the current project as an alternative to architectures defined previously in the current chapter.

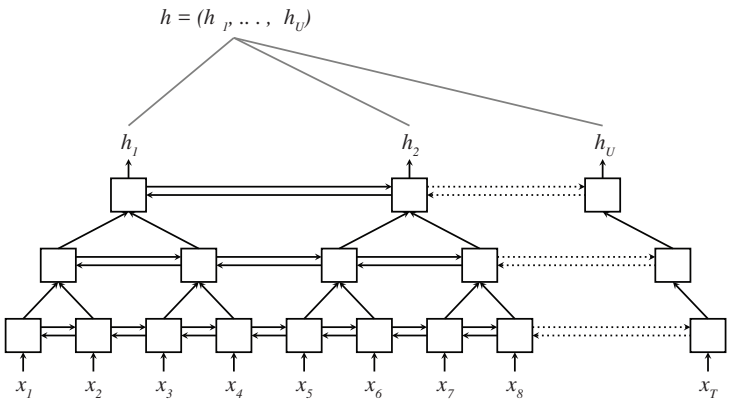


Figure 14: The Pyramidal RNN architecture resembles Lerdahl and Jackendoff's Time-Span reduction structure. [Chan et al., 2016]

The idea of hierarchical RNNs is not new [El Hihi and Bengio, 1996]. The implementation that is used in the current paper is fairly recent though [Chan et al., 2016]. It was originally applied to the problem of audio transcription which shares similarities with the current project. Chan et al. implemented the pyramidal time-span reduction mechanism because their model was failing to converge. Just like the notated music used in the current project, input speech signal can be hundreds to thousands of frames long. This is in contrast to language sentences, which are tens of words at most.

The Pyramidal architecture is fairly simple and rather crude compared to the more sophisticated chunking proposed by Lerdahl and Jackendoff. However, Pyramidal RNN is good enough to give a directional answer if continued efforts are warranted. The design should be able to provide a level of abstraction focusing the attention of the network on the meaning of a collection of events, as opposed to individual events. Mathematically, the idea behind Pyramidal RNNs is to half the time resolution with each pyramidal layer. So, a 3-layer stack reduces the dimension $2^3 = 8$-fold. The dimensionality is reduced by concatenating the output from lower layers before feeding it into higher levels. The design additionally decreases the computational complexity boosting learning and inference times.

7.4 Transformer

Finally, a model called Transformer is tested. It originates from Google research and it has shaken up the status quo bringing major improvements to machine translation results. The specifics on the workings of the model are complex and beyond the scope of the current report. It suffices to say that the model is based on a set of convolutional, recurrent and feedforward neural networks that completely change the traditional wirings of the traditional relationship between encoder and decoder. One of the biggest advantages of the model has been its low computational footprint, allowing much faster training than other models used previously [Vaswani et al., 2017]. A good annotated explanation of the original paper is available [4] [Rush, 2018].

[4] http://nlp.seas.harvard.edu/2018/04/03/attention.html

Part V

Support Infrastructure

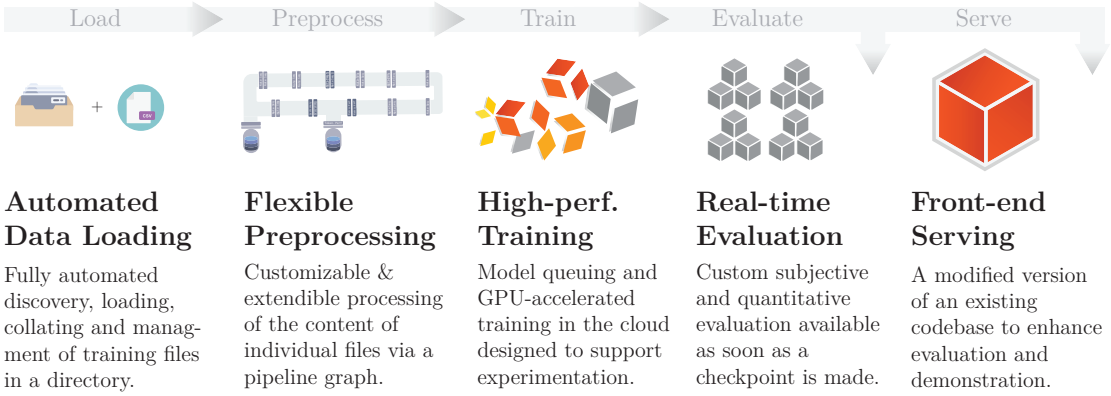


Figure 15: A high-level overview of the process supported by the software architecture designed and implemented for the current research.

The current project began with the ambitious goal to run systematically a large number of experiments. The timeline of less than 8 weeks for designing and implementing the infrastructure was challenging. This section outlines some of the main challenges.

When one hears a dataset of 200 pop songs, one may think this is not many. Not quite. The original songs are stored as over 12,000 separate files amounting to 10.4 Gigabytes in a proprietary and heavyweight file format. Each song is available in at least two *arrangements*, and each arrangement is split into many *segments* (whole song, chorus, verse, intro, outro, etc.), and each of those is stored in three separate files, one for the left-hand part, another for the right-hand part, and a third for both hands. The data was not immediately available at the beginning of the project. Instead, the entire dataset was gradually becoming available via a shared directory.

The setup above necessitated the building of a flexible and robust system for handling a live stream of incoming data. The system for loading, collating, preprocessing, training and evaluation was incrementally built out and unit tested. Once the entire dataset was available, all experiments were ran from a queue in the cloud. The approach avoided the bottle-neck of data unavailability. The training was done on a high-performance industry-grade GPU-powered cluster of virtual machines. The final solution is based on more than a dozen Python libraries, some of which were forked and customized extensively (i.e. Google Magenta[5]).

[5]https://magenta.tensorflow.org

Automated Data Collating

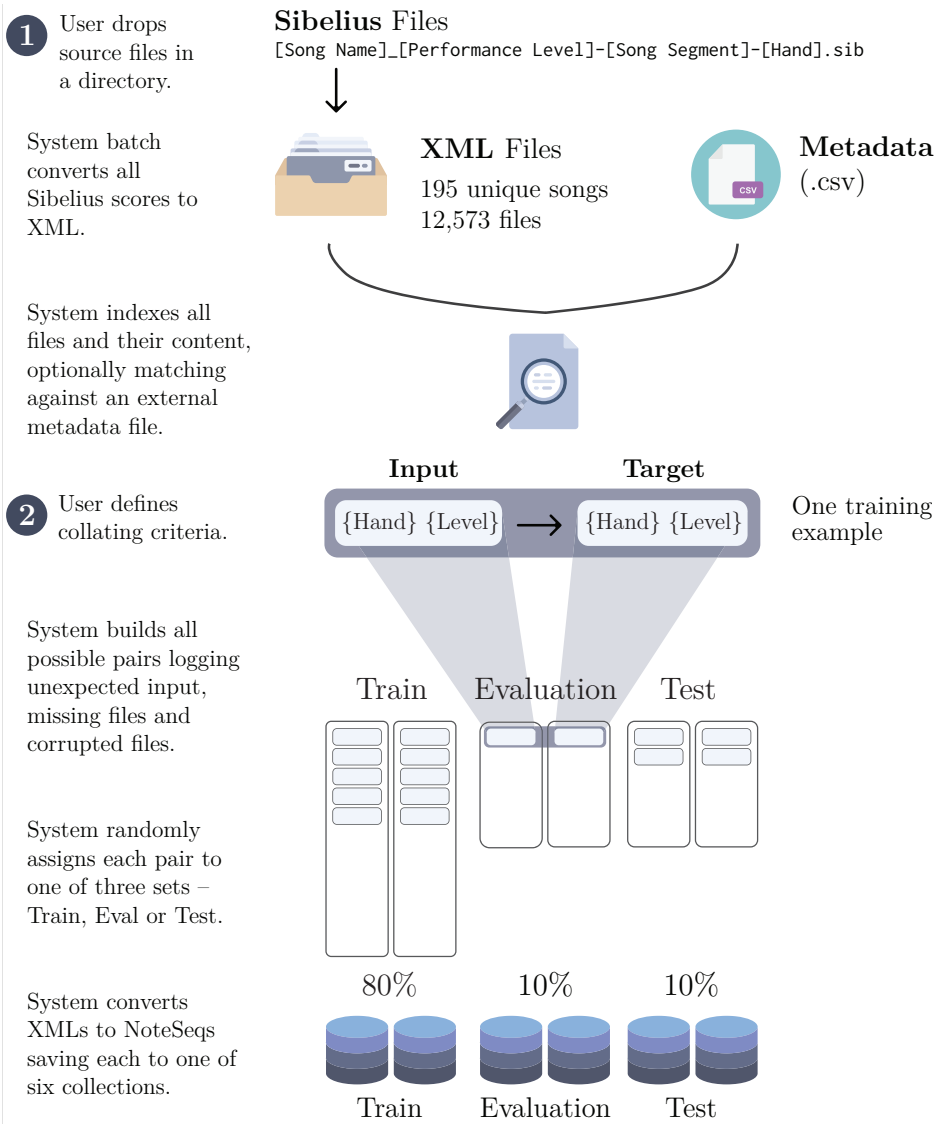


Figure 16: Diagram showing the step of loading and collating the training data from a source directory. The collating module searches recursively files in a directory, recognizes the filename pattern, optionally references an external file with metadata. The module builds an index with the collected information. Once the index of is constructed, the system organizes the training pairs into input and target according to user-specified criteria, and splits the collated pairs into three datasets: train, evaluation and test. Image by the author. Some icons by flaticon.com.

Figure 16 details the process of getting the original 12,000 source files from a shared directory to a neatly collated collection of 6 datasets – training, evaluation, and test. A data administrator needs to take care of only two actions:

1. Point the system to a source directory
2. Define collating criteria

The second step is important because it allows easy reorganization of the source data into multiple version which are used for different experiments. When choosing how to collate files, a user can define a list of multiple criteria for the inputs and the targets – specifying the hand (left for accompaniment, right for melody or both), level (beginner, intermediate, or advanced) and segment type. Optionally, the system can reference an external metadata file holding information for the tonic, key, time signature, genre, name and artist of each song. The system performs error-checking, keeps a log and notifies the data administrator of any exceptions.

Flexible Preprocessing

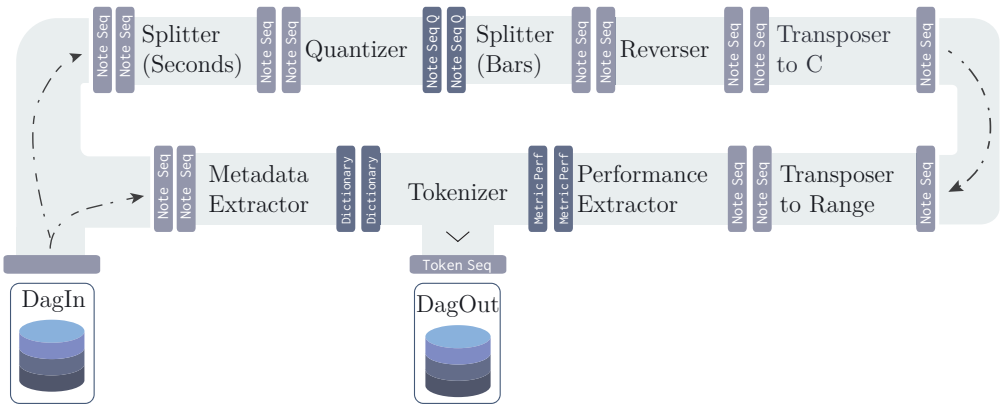


Figure 17: Illustration of one possible configuration of the implemented DAG (Directed Acyclical Graph) for data preprocessing. All of the pipes used in the current project are shown. Notice that the graph splits at the onset into two branches which are later merged before right saving the processed output to disk. The longer branch does the legwork of musical transformations, while the shorter branch is used to augment the music with metadata from an external file.

The second key piece in getting the data ready for training is the preprocessing of the content of individual files. While the file collating module performs operations on files in a directory, the preprocessing pipelines module alters the musical content of individual files.

The focus during preprocessing is on flexibility. Pipelines can be removed, inserted and connected in different order in a DAG, a Directed Acyclical Graph. Each pipeline takes input of pre-defined data object(s), performs a transformation and outputs pre-defined data object(s). The pipeline graph can be customized in any number of ways as long as the resulting graph is acyclical and the output of a pipeline is the same data type of the input of the next pipeline. The high-level skeleton of the pipeline graph is borrowed from Google's implementation for the Magenta project[6]. However, almost all pipeline definitions, transformation methods, parsings and even the specific data types are customized for the current project.

Splitter Splitter takes a `NoteSequence` and chops it on regular intervals. There are two version of **Splitter**, one for splitting by seconds, and another for splitting by bars. A `NoteSequence` must be quantized first if splitting by bars, and a `NoteSequence` must not be quantized if splitting by seconds.

Quantizer Quantizer transform a `NoteSequence` from real time to metrical time with an appropriately chosen value of steps per quarter note.

Reverser Reverser takes a `NoteSequence` and reverses the order of all notes, rests and other events. The pipeline takes one sequence and return two. This is a pipeline for data augmentation.

Transposer to C Transposer to C takes a `NoteSequence` and transposes it to middle C. This is pipeline for data standardization.

Transposer to Range Transposer to Range takes a `NoteSequence` and transposes it to a range of values. For example, transposition to `[-12, 12]` transposes the input to all keys an octave lower and octave higher from the current key of the composition. The pipeline takes one melodic sequence and outputs one or many sequences. This is a pipeline for data augmentation.

Performance Extractor Performance Extractor converts a `NoteSequence` to a `MetricPerformance`. See data representation in Part III: Data for details on data types. This is a pipeline for data type change.

[6] https://github.com/tensorflow/magenta/tree/master/magenta/pipelines

Metadata Extractor Metadata Extractor fetches relevant metadata from an external file for a given `NoteSequence`. This is a pipeline for data augmentation.

Tokenizer Tokenizer takes either only a musical input (as a `MetricPerformance` data type), or music with a dictionary with metadata as `key:value` pairs. Tokenizer transforms `MetricPerformance` to `TokenSequence`. Again, reference Part III: Data for details on the two data types. This is a pipeline for data type change.

High-performance Training

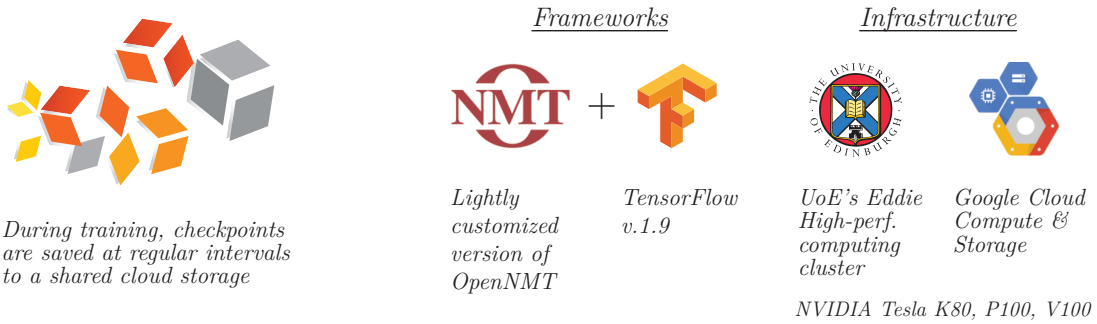


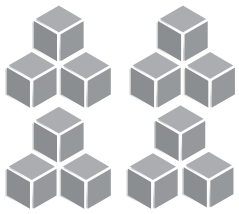
Figure 18: Checkpoints are saved on regular intervals for all models in the training queue. OpenNMT, an abstraction around TensorFlow, is the main framework used in the course of the project. Training is done in the cloud using GPU-accelerated virtual machines with four NVIDIA Tesla K80, two P100 and one V100.

OpenNMT, the Open-Source Toolkit for Neural Machine Translation [Klein et al., 2017] is the main framework used to support fast and iterative experimentation. OpenNMT[7] has implementations for LuaTorch, PyTorch and TensorFlow, the last of which was used in the current project. OpenNMT was easily extended with music-specific evaluation metrics (see Part VI: Experiments). The clearly written and modular code was altered to enable evaluation on every checkpoint, a functionality which was not available out-of-the-box. Getting started was straightforward with good documentation and active community. Guillaume Klein,[8] the principal researcher of the project, responded in hours to issues submitted to the project's GitHub repository, including to a small bug pull request discovered and fixed in the course of the current project [`Pull #166`].[9]

[7] http://opennmt.net/
[8] https://www.linkedin.com/in/guillaumekln/
[9] https://github.com/OpenNMT/OpenNMT-tf/pull/166

OpenNMT is similar The University of Edinburgh's *Nematus* project[10] [Sennrich et al., 2017]. However, OpenNMT was preferred due to offering out-of-the-box implementation of a Pyramidal RNN Encoder. Additionally, OpenNMT implements an external memory sharing-system and data-parallelism for multi-GPU training which provide savings of GPU RAM memory usage and significant speed-up per epoch in multi-GPU setting. The RAM savings are crucial for the current project because music sequences are typically hundreds of events long, compared to just tens of words in a sentence in natural language setting. Long input and output sequences cause the creation of massive matrices and during training, the GPU runs out of memory exiting with an error. One solution is to reduce the batch size, but even with a reduced batch size, the training would have been impossible without OpenNMT's RAM usage optimizations.

Real-time Evaluation



All checkpoints of all trained models are available and can be selectively loaded for evaluation

TensorBoard

TensorBoard server on a remote Virtual Machine

+ Loss Monitoring
+ Embeddings Vis.
+ Custom Metics

Local Machine

Local server with Jupyter Notebooks

+ Audio Synthesizer
+ Pianoroll Visualization
+ Attention Visualization
+ Custom Metrics

The cubes on the left is derived from an infographic located at tensorflow.org/serving. All logos are property of their respective owners.

Figure 19: All checkpoints are evaluated in real time and are available on demand for later evaluation.

During training, TensorFlow saves *checkpoint* and *event* files, while Open-NMT runs *inference*, saving predictions to text files. All files are store in a single cloud-based location and are accessed on-demand from various machines. TensorFlow's event files are useful to analyze plots of various quantitative metrics in a browser with TensorBoard[11]. Precision, recall, F-1 score and custom music-specific metrics, as well PCA and T-SNE projection of the trained embeddings are available (Fig. 27, Fig. 28 and Fig. 29).

Unfortunately, the standard TensorBoard implementation is not well-equipped for niche areas like music and additional tools were developed. Jupyter Notebooks running on a local machine were developed to connect remotely to the cloud storage accessing

[10] https://github.com/EdinburghNLP/nematus
[11] https://www.tensorflow.org/guide/summaries_and_tensorboard

model checkpoints and cached inference files. The cached files provide quick access to additional insights which are not available with TensorBoard. Most useful proved to be two interactive tools for evaluation: an audio synthesizer which can load data from the cached inference files, as well as a pianoroll visualization which uses the same data (Fig. 30). Another important local implementation allows a visualization of a trained model's activation layer which is recoverable from checkpoints (Fig. 32).

Front-end Serving

The red cube on the left is derived from an infographic located at tensorflow.org/serving. All logos are property of their respective owners. Firewall icon by flaticon.com. Browser mockup template by freepik.com. Infographic by the author.

Figure 20: The best model is available interactively in a browser via an interface originally developed by Yotam Mann for Google Magenta and repurposed for the current project.

Music is made to be heard. Given the fantastic results which the trained model achieved, it would have been pity to describe music with words, charts and tables.

To present the fruits of the current project, a Google project from two years ago was revived and repurposed. A.I. Duet Yotam Mann[12] is an experiment allowing a computer and a human to play interchangeably on a virtual piano keyboard in a web browser. A user presses a few keys and hears to what comes back (see Fig. 21 for a screenshot of the interface).

It is important to highlight a few difference between Yotam Mann's A.I. Duet and the current project. Even though both use the same beautfully sleek design, what happens under the hood is very different. Backstage, A.I. Duet is based on a Google Magenta project called Melody RNN[13] which takes a few monophonic pitches and always predicts the next one. After the first prediction, the model takes the original set along with with the newly predicted pitch and predicts one more pitch. The process of *continuation*, however, is monophonic (only one note sounds at a time), it has no global musical

[12]https://experiments.withgoogle.com/ai-duet
[13]https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn/

structures and the exchanges are very short on average, clipped at 5-6 seconds. On the other hand, the current project, A.I. Enhancer, does not just continue the user's input but generates a rendition it supported by chords and often a slight tonal or rhythmic improvisation. A.I. Enhancer is showing understanding of the melody and adds to its richness. Additionally, Enhancer corrects imperfections – such as inaccurate timing in the user input. Lastly, Enhancer can easily handle input of length up to 30 seconds, and even longer albeit with some degradation of the quality of the prediction. Further details on the evaluation and abilities of A.I. Enhancer are presented in Part VII: Results.

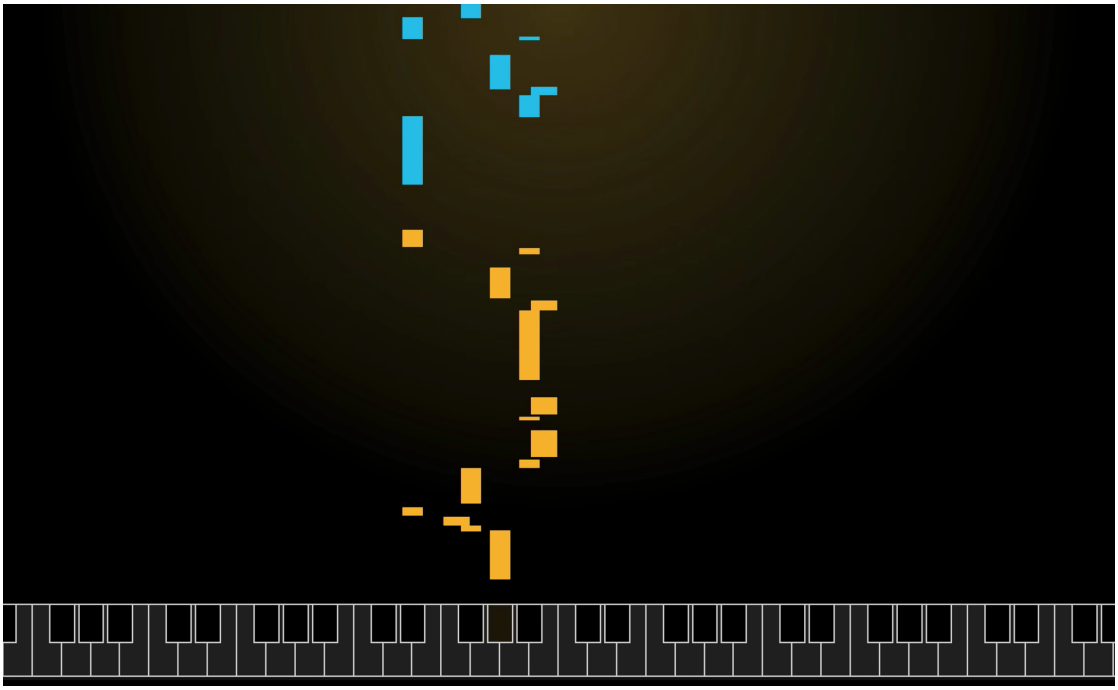


Figure 21: Screenshot of the interactive frontend.

Part VI
Experiments

The current chapter shows the list of experiments which were planned from the onset. All experiments can be roughly grouped into two groups. The first benchmarks the performance of various architectures types and hyper-parameter choices. The second group, which is larger, varies the way training pairs are collated and preprocessed (varying sequence length, use of augmentation, use of metadata, and level of difficulty). Twenty-six different datasets were constructed from the original dataset, each named after a song bird.

8 Pre-Baseline

Pre-Baseline experiment are run to ensure the datasets are constructed correctly, data flow correctly through the model, the network is constructed properly, it learns, results are sensible, the architecture is fit for the purpose, and hyper-parameters are reasonable. Results from those experiments are not reported.

9 Baseline

The Baseline experiments follow a grid-search setup to find an architecture and dataset slicing which are assumed as a baseline. Subsequent experiments are measured against the baseline.

9.1 Model & Hyperparameters

Embedd. Size	No. Layers	No. Units	Encoder	Decoder	Learn. Rate	L2 Regular.
64, 512	2, 4	512, 1024	Plain RNN, Bidirectional RNN	Plain RNN, RNN + Attention	0.05, 0.005 0.001	1e-2, 1e-4 1e-10, 1e-100

Table 2: The baseline model and hyperparameters are chosen after a grid search of the possible shown values above.

The first set of experiments varies the embedding size, the depth of the neural network, the number of computational units in each layer, the type of encoder and decoder,

as well as the learning rate and the L-2 regularization value. LSTM and GRU neurons are evaluated. The encoder and decoder networks are symmetric, always with the same number of layers and neurons. Note that those choices are fairly standard, known to produce state-of-the-art results in a wide varied of cases. The goal here is to spend as little time as possible getting reasonable baseline results, and continue to more interesting experiments.

9.2 Data Standardization

A typical standardization in music informatics is to transpose all training examples to the same key to prevent confusion of the model. This is especially important because in the current data one and the same song is sometimes arranged in a different key across levels of difficulty. For example, Beyonce's *Best Thing I Never Had* is **F** in beginner intermediate and **G flat** in advanced level. Clearly, the model has no way to know of such arbitrary choices and probably they will be confusing. The current experiment aims to give an answer how robust the trained model is to such anomalies. The hypothesis is that the Key Accuracy metric will suffer significantly, but will other metrics like Precision and Recall, Tempo and Note Density suffer too?

	All to C?	No. Training Examples
I	No	2543
II	Yes	2543

Figure 22: Transposing all tracks to key C is a common standardization technique. How much does it really matter?

9.3 Data Augmentation

These experiments aim to find what augmentations help the model achieve higher metrics. All augmentations increase the number of training examples in some way. Before augmenting, the dataset will be first transpose to C, if the results from the previous experiment confirm the hypothesis that this standardization is important.

	Augmentation	No. Examples
I	None	2543
II	Reverse	5086
III	Transpose [-12, 12]	61032

Figure 23: Data is king in machine learning, and one may to make more data is through augmentation. Which augmentation will boost performance the most?

10 Core Experiments

10.1 Conditioning on Metadata

This set of three experiments investigates whether metadata boosts performance. Both the encoder and decoder are conditioned by prepending metadata tokens at the first time step. This sets the internal state of the LSTM neurons.

Arguably, the chosen method for conditioning is not the best from a theoretical standpoint because the network may forger the conditioning with very long sequences. A more correct approach is to *directly* set the first hidden state by, first, applying an affine transformation to get the metadata embeddings to the same shape as the LSTM's hidden state, and, second, add the transformed embedding to the hidden state [Karpathy and Fei-Fei, 2015, Vinyals et al., 2015b].

Tokens	Exp. ID
[Segment. Genre.]	`meta_1`
[Key. Mode. Time Sign.]	`meta_2`
[Segment. Genre. Key. Mode. Time Sign.]	`meta_3`

Figure 24: We can tell the trained model explicitly what is being encoded. Will metadata help, or can the model figure it all out by itself?

That said, the first method is faster to implement because it does not require redesigning of the existing training infrastructure. The first approach is *good enough* to give a directional approach whether metadata helps and approximately how much it helps. If the results are promising, we can return and do a more theoretically refined implementation.

10.2 Varying Sequence Length

This set of experiment is designed to determine what is the maximum sequence length that the model can sensibly reproduce. The default in the baseline is a *segment*-length, which is a variable chunking of the music content according in segments which semantically make sense and are audibly perceived differently by people (i.e. chorus vs. verse). The results will be certainly interesting but probably not conclusive. This is because the number of training examples is inversely proportional to the sequence length. This is because the dataset size is fixed and the longer the pieces in which we slice a song, the fewer pieces there are.

Sequence Length	Avg. Length In Measures	Avg. Length In Seconds	No. Training Examples	Exp. ID
1 Bar	1 bar	2 sec.	59,349	`len_1`
2 Bars	2 bars	4 sec.	29,819	`len_2`
5 Bars	5 bars	10 sec.	12,127	`len_5`
10 Bars	10 bars	20 sec.	62,25	`len_10`
Segment	13 bars	25 sec.	5086	`baseline`
20 Bars	20 bars	40 sec.	3,271	`len_20`
60 Bars	60 bars	2 min.	1,281	`len_60`
Full Song	100 bars	3 min 35 sec	200	`len_f`

Table 3: Recurrent neural networks, even with LSTM and GRU neurons, are infamous for forgetting the oldest events. How many seconds of music is too much?

10.3 Harmonizing Varyingly Complex Melodies

The current set of three experiments aims to determine at what level adding an accompaniment the model struggles most, and whether the difficulty increases gradually or one level is more different than the other two. Notice that the model is not asked to increase the performing complexity of a song but only to add the left hand part given the right hand part.

Level	No. Examples	Exp. ID
Beginner	2,174	`acc_1`
Intermediate	2,202	`acc_2`
Advanced	1,470	`acc_3`

Figure 25: The set of experiments above challenges the trained model harmonize melodies of different performing levels. Is advanced really that much harder than beginner?

10.4 Increasing Melodic Complexity

The current experiment ignores the accompaniment in the left hand and only uses tracks which carry the melody line. The model is challenged to increase the expressiveness of a melody from beginner to intermediate level, from intermediate to advanced level, and from from beginner to advanced level. Table 4 highlights a few important differences between the three levels which the trained model must learn to transform between.

	Min. Note	Chords
Advanced	1/16	Yes
Intermediate	1/8	No
Easy	1/8	No

Table 4: Differences in the melody complexity between beginner, intermediate and advanced levels.

Challenge	No. Examples	Exp. ID
Beg –> Int	2,174	`express_1`
Int –> Adv	1,470	`express_2`
Beg –> Adv	1,442	`express_3`

Table 5: Expressiveness. The set of three experiments above tests if the trained model can increase the performing complexity of a melody. To do well, the model must make sensible notes out of thin air.

11 Evaluation Metrics

Evaluation is one of the most important components in research. In many research areas there is often a gold standard (i.e. the BLEU score in Machine Translation). Unfortunately, this is not the case in music. The present project felt the need for quantitative measure early on as it is unfeasible to inspect enough random samples qualitatively for each of hundreds of experiments that were ran, and during each of the hundreds of checkpoints during training.

The section below shows two groups of metrics – the first standard statistical measures such as precision, and recall which have proven highly informative. The second part is the most interesting group of metrics, many of which were designed by the author and have not been used as a collection in other research (though certainly the metrics have been used individually).

Scores are calculated on the evaluation dataset (10% of the entire available dataset) every time a new checkpoint is saved during training.

11.1 Statistical

To evaluate parts of the model, we will use precision and recall, and to compare the models as a whole in picking the best one, we will use BLEU.

BLEU BLEU, the Bilingual Evaluation Understudy Score, was originally developed by IBM researchers for the US Department of Defense as a way to automate the evaluation of translation systems [Papineni et al., 2002]. BLEU takes values between 1 and 100, with 100 indicating complete match between a prediction and reference output. The metric is claimed to correlate highly with human judgment [Coughlin, 2003], but even professional human translators do not achieve perfect scores. Current machine translation systems achieve a score of high 30s [Wu et al., 2016]. Even though the metric is not a benchmark

in music, the fact that is a precision-based score makes it universal and informative to the current project. Without over-relying on BLEU initially, the current project has established that there is a high correlation between a high BLEU score and the ability of a model to produce a good accompaniment.

ROUGE Recall-Oriented Understudy for Gisting Evaluation has been the new kid in the block and it has been an attempt to replace BLEU by bringing in the entire suite of Precision, Recall and F-1 [Lin, 2004]. Three types of ROUGE were used in the current project: ROUGE-1 and ROUGE-2, measuring the Precision, Recall and F-1 scores of unigrams and bigrams, respectively, and ROUGE-L which calculates the three scores for the Longest Common Subsequence matching sequence. In practice, the current study found best scores for F-1, followed by F-L and trailed by F-2. In the course of the current project, the author submitted a pull request for improvement of the current Python ROUGE implementation, [`Issues #14`][14].

An important difference between BLEU and ROUGE is that the BLEU score implements a *brevity penalty* which penalizes unnecessarily short predictions to discourage gaming of the system. That said, the ROUGE score is more straightforward to interpret.

A note of caution must be raised that neither BLEU, not ROUGE measure the *fluency* of the predictions but only the *adequacy*. In other words, predictions will get a high score as long as the right n-grams events are recalled and they are correctly recalled. However, the metrics simply count n-grams and the n-grams can be shuffled in any order. A higher BLEU score has been shown to be "neither necessary nor sufficient" for better translation. [Callison-Burch et al., 2006].

11.2 Music-specific

Music-specific metrics are reported as both a Score and a Distance.

Score is the average score of all predicted sequences after evaluating each sequence by itself in isolation:

$$\text{Score} = \frac{1}{N} \sum_{Y} metric(y_n)$$

Distance is the average distance between the scores of all predicted sequences relative to the scores of all respective target sequences:

[14] https://github.com/pltrdy/rouge/issues/14

$$\textsc{Distance} = \frac{1}{N} \sum_{Y,\hat{Y}} |metric(y_n) - metric(\hat{y_n})|,$$

where Y is the set of predicted sequences, and $\hat{Y}$ is the set of target sequences.

Correlation Coefficient (Score & Distance) The correlation coefficient is a very useful metric from the music21 package.[15] The metric analyzes a performance and produces a Pearson coefficient score indicating how well the key of the composition fits the profile of other composition in the same key. The range of possible values is between -1 and 1, with high negative values indicating high negative correlation, high positive values indicating high positive correlation and 0 indicating no correlation. High values are better.

Tonal Certainty (Score & Distance) Tonal Certainty is another metric implemented in the music21 package. It is derived from the value of the correlation coefficient. Lower tonal certainty values indicate ambiguity because the composition uses pitches belonging to different scales. The metric is a useful proxy measuring how well the trained model has learned the concept of tonality.

Key Accuracy [%] Key Accuracy shows the percentage of predicted sequences which are built around the same tonic note as the target sequence (irrespectively whether the key is major or minor). The range is from 0% to 100% and higher values are better.

Mode Accuracy (Score) [%] Mode Accuracy shows the percentage of predicted sequences which are the same mode (major or minor) as the target sequence. The range is from 0% to 100% and higher values are better.

Duration (Distance) [Seconds] Duration is the difference between the duration in seconds of the predicted music sequence against the target seconds. Ideally, we would want to minimize the duration distance to 0, which would indicate that the model has a perfect sense how long the desired output is given an input of any length.

Note Density (Ratio) Note Density is measured as the ratio between the the number of events in the predicted sequence over the number of tokens in the target sequence. Note Density is related and often correlated with Duration Distance but provides useful

[15] http://web.mit.edu/music21/doc/

guidance which Duration Distance does not. Even if two sequences are of the same duration, one may be simpled than the other – for example the predicted melody may only a single voice, while the target melody has several voices and more notes respectively. In this case, even if the duration distance is minimized, the note density ratio will be lower than the best possible value. Other than polyphony (fewer `ON` and `OFF` events), another reason why note density may be lower is if the rhythm complexity of predicted sequence is lower than the target sequence (fewer `SHIFT` events).

Tempo (Score & Distance) [BPM] Tempo measures the beats per minute of the predicted music sequence and the distance from the target sequence. The score calculates empirically the probability of the most likely tempo by detecting rhythmic events, and generating tempo hypothesis in different metric scenarios [Dixon, 2001]. The metric sometime halves or doubles the tempo but is generally robust. The main advantage of the used implementation[16] is that it does not require prior information on genre, meter of tempo of the input (which are not available for inferred sequences).

[16] https://github.com/craffel/pretty-midi

Part VII
Results

A massive number of experiments were ran for this report – a few hundred grid-search tests to establish the baseline architecture and hyperparameters, and tens of additional experiments to report the final results below. Loss, BLEU score, nine types of Rouge scores (ROUGE-[1,2,L] Precision, Recall and F-1) and ten music-specific scores were recorded for each checkpoint for each experiment. Naturally, a balance must be found between giving the reader as much of a complete picture as possible without overwhelming.

All experiments were run for a sufficient number of epochs and were stopped not earlier than before the loss began to flatten out. Most often the last step is reported. However, where experiments were run for too long and the model began to overfit, an earlier step with a lower loss on the evaluation dataset is preferred.

Different experiments were run for a different number of steps. Some models converged faster than others. For example, experiment `len_1`, which was trained on examples of length 1 bar, began overfitting after 4,635 steps, which equals roughly 5 epochs. On the other hand, experiment `len_5`, which was trained on examples shorter than 5 bars, ran for 30 epochs and the loss was still decreasing without overfitting (though, the rate of decline was fairly flat by that point).

Note on how tabular results are reported:

- The first column shows the manipulated variable. All other variables – from model architecture, to hyperparameters to dataset slicing are held constant.
- Tables have a variable number of columns and a judgment is made which columns are included. Generally, metrics which vary little across rows are hidden from view for clarity and ease of comprehension.
- The last column of each table shows a unique identifier of the experiment which can be used to reference the look up the full results. The full results will be published on the GitHub repository of the project,[17] and are available from the author on request.

[17] https://github.com/cholakov/UoE-dissertation

- Only ROUGE-L is reported as a Precision, Recall and F-1 score. This is because R-1, R-2 and R-L were found to be highly correlated, while R-L provides a good trade-off between the simpler unigram evaluator ROUGE-1, and the more challenging ROUGE-2. In addition, ROUGE-L is more flexible as it automatically identifies and takes into account the longest co-occurring subsequence.

12 Findings

12.1 Standardization is Important

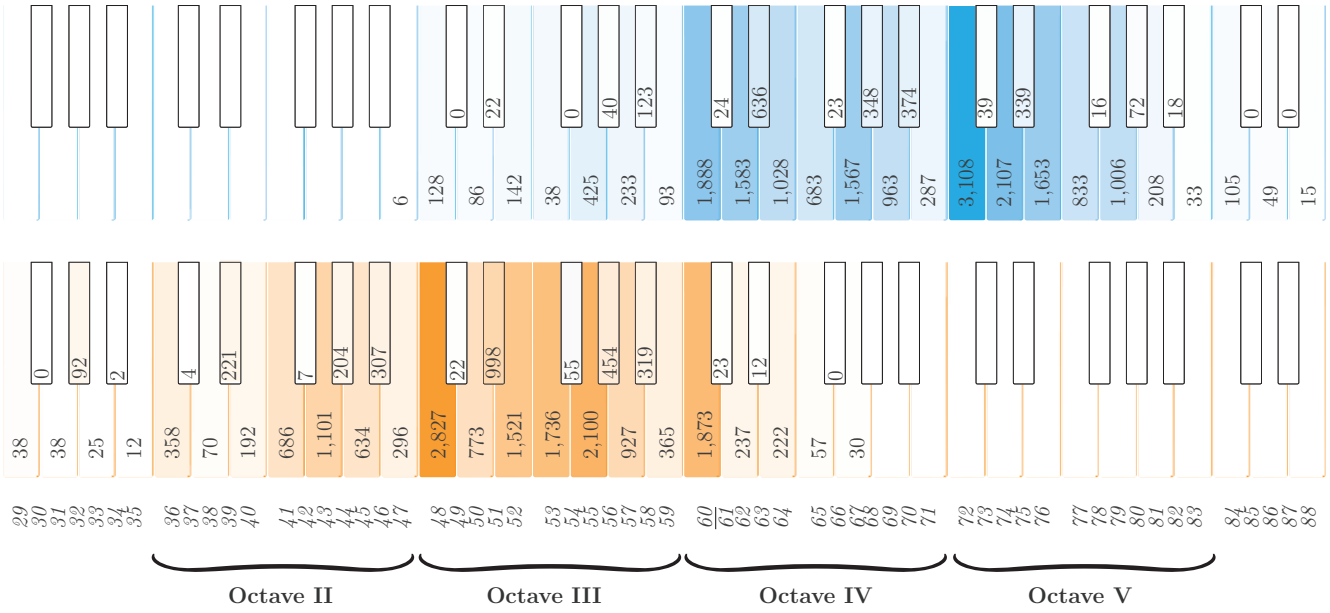


Figure 26: A histogram visualizing the frequency at which different pitches appear in the training dataset *after* standardization. The top keyboard (blue) shows the distribution of pitches in the part played by the right hand (the melody), while the bottom keyboard (orange) shows the distribution of pitches in the left-hand part (the accompaniment). The numbers on the keys indicate counts. All songs have been first standardized to the key of C and no augmentations have been applied. The row of numbers below the bottom keyboard indicates the MIDI number which corresponds to a given pitch. The underlined number 60 is middle C on a standard piano keyboard.

The present set of two experiments confirms our initial hypothesis. Standardization to the same key matters and there is a good reason why this is a common practice in music informatics. Due to the nature of the training data, if songs are not transposed to the same base key, the source and the target training pairs are occasionally in different keys. Consequently, the model is confused. The Key Accuracy score jumps from mediocre 14% to close to 50% after standardization. With 8 percentage points better, the model

All to C?	ROUGE-L F-1	ROUGE-L Prec.	ROUGE-L Recall	Key Acc.	Mode Acc.	Tonal Cert.	Tonal Cert. Dist.
Yes	22%	81%	21%	49%	68%	1.20	0.27
No	22%	73%	21%	14%	60%	1.26	0.36

Table 6: Standardizing all tracks to middle C drives a significantly better Key Accuracy score.

outputs music in the correct mode (major or minor). Given this was one of the first experiments, the expected results were a good sanity check confirming that the custom implementation of the accuracy scores works correctly. Interestingly, while precision is boosted, the recall of events remains unchanged between regardless of standardization.

12.2 Dataset Size Matters. A lot.

Next comes one of the most eye-popping revelations from the current project: good augmentation makes or breaks. Two augmentation methods were evaluated:

- Flipping the music sequences in reverse, a general approach suggested by [Sutskever et al., 2014].
- Performing a music-specific augmentation which is unavailable in machine translation – tonal transposition.

The first approach helped little, and the second made all the difference.

Sequence Reversal Is Hyped In general, sequence reversal helped with *stability* during training but the final results were not improved. Examining metrics during training showed that song reversing reduces zig-zagging of the plots and drives a steadier decrease. That said, experiments converge eventually to the same values with and without reversing. The scores from reversing the music sequence are not reported here because they are almost exactly the same as the `baseline` scores shown in later tables.

Transposition is Amazing Augmenting the dataset by transposition an octave lower and an octave higher improves performance by a staggering amount. The loss decreases from 1.83 to 1.28, the BLEU score jumps from 5.40 to 13.92. Event recall (ROUGE-L) doubles from 21% to 50% while precision drops slightly by 7 percentage points. The most impressive improvement is seen in the metric most difficult to get right: recall, and specifically bigram recall, which improves from 21% to 50%, a factor of two.

The improvement can be attributed to the fact that transposing all tracks an octave up and down increases the number of training datapoints 24 times. Each music sequence is transposed from its home key to a range [-12, +12], after all tracks are first standardized to middle C (in line with the earlier finding that standardization is important).

While the larger dataset slows down training, the trained model does not need nearly as many iterations (epochs) to achieve a better loss and scores than the non-augmented dataset. After just 5 epochs, the augmented dataset helps drive a staggering improvement, while the non-augmented dataset is trained for 30 epochs.

Transposition	No. Train. Pairs	Trained for	BLEU	R-L F-1	R-L Prec.	R-L Recall	Tonal Cert.	Tonal Cert. Dist.	Tempo	Tempo Dist.	Density
None	5,086	20 epochs 1,589 steps	5.4	23%	82%	21%	1.20	0.26	215	43	0.93
[-12, 12]	122,064	5 epochs 10,076 steps	13.92	55%	75%	50%	1.23	0.35	186	32	0.66

Table 7: Dataset augmentation by transposition drives big improvements, up to several times better for some metrics.

Additionally, the augmented data helps the composition of arrangements which better match the beats-per-minute and the length of the target sequence. The tonal certainty score trends a little higher. This is to indicate that with more data, the model learns better the principles of tonality and centers the composition around a tonic note which predicates the other notes and chords. Strangely, with more data, the tonal distance increases. As a reminder, the tonal distance is measured as the difference in tonal certainty of the target sequence and the predicted sequence. The phenomenon can be interpreted to mean that, unfortunately, with additional data the model becomes conservative and produces output which more strictly conforms to more rigid rules, as opposed to taking small creative deviations. One explanation why the Tonal Certainty and the Tonal Certainty Distance go in different direction is the presence of some jazz music in our corpus. Notably, jazz includes many of Western music' tonal characteristic but it also bends a lot of rules to please the listener's ears or respond to an impulse of the musician's hand, neither of which our recurrent neural model can comprehend.

Note Despite the good results, to reduce training time, a decision was made not to use augmentation in running most of the experiments reported next. Augmentation is used in training the final single best model.

12.3 Metadata Helps. Somewhat

Metadata	ROUGE-L F-1	ROUGE-L Prec.	ROUGE-L Recall	Tonal Cert.	Tonal Cert. Dist.	Exp. ID
None	22%	81%	21%	1.20	0.27	`baseline`
[segment, genre]	27%	79%	25%	1.19	0.26	`meta_1`
[mode, key, time_sig]	28%	80%	26%	1.16	0.29	`meta_2`
[segment, genre, mode, key, time_sig]	30%	74%	27%	1.14	0.27	`meta_3`

Table 8: Metadata. Conditioning the encoding and decoding on metadata improves results, though not massively.

The effects of metadata on performance are more difficult to judge. On one hand, conditioning the encoding and decoding on metadata has no downside. On the other hand, the improvement manifests itself only in a better recall score, which also drives a higher F-1 score. Clearly, this is not insignificant because the F-1 score, a single combined score combining both Precision and Recall, improves from 22% in `baseline` to close to 30% with the use of metadata. Music-specific metrics generally do not change. Tonal Certainty Score and Distance are the biggest movers, but even they move little. Tonal certainty decreases a little, while distance hovers around 0.27. The result can mean the model becomes more aware of the subtleties of different genres and does not dogmatically force a fit to the most theoretically right tonal scale. This is a good thing. But is it enough to warrant the use of metadata as opposed to working with pure music?

Metadata is difficult to obtain and having as good metadata as the current project has is more an exception than the norm in music in machine learning. The bottom line is that metadata on the mode, key, genre and time signature is great to have but not essential for good results. The biggest advantage of having metadata is that it allows us to understand how the trained model works under the hood, as a few beautiful visualizations in the following subsection reveal.

12.4 Easy To Learn Fundamental Music Concepts

Table 27 shows that in less than 3,000 training steps, a model learns to use pitches with a frequency typical to other compositions of the same tonic (Tonal Certainty Score). The model gets the right key even earlier, with roughly 2,000 steps. Similarly fast the model optimizes duration, beat, mode accuracy and note density. Meanwhile, precision, recall and the combined F-1 score do not reach their best values until the 10,000th step. The

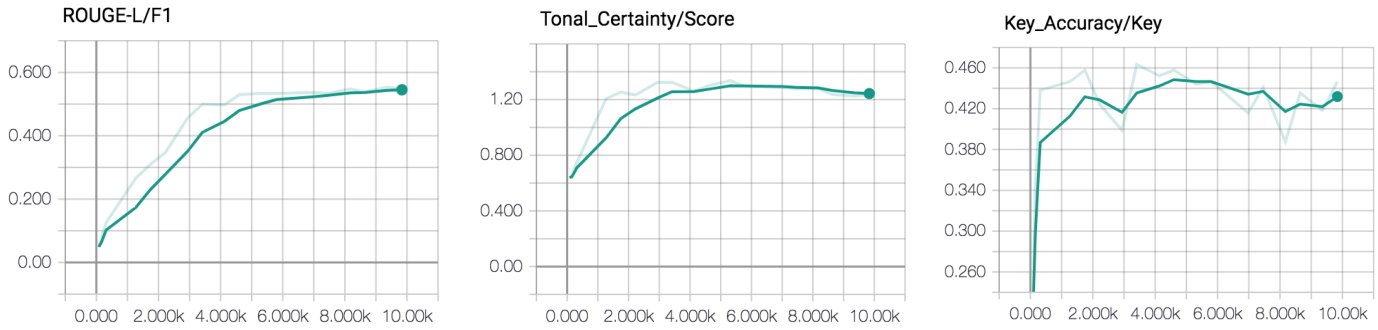


Figure 27: Basic music notions are relatively easy to learn. Screenshots from TensorBoard show that the trained model learns faster to stay in the right tonal scale (Key Accuracy) and use sensibly pitches from that scale (Tonal Certainty) than to model the melodic content semantically (F-1 score). The shown plots are from training of experiment with ID `prepro_3`. The darker-colored line is a smoothing of the original line, slightly faded-out, by a factor of 0.6.

results can be interpreted to mean that our sequence-to-sequence model picks up the fundamentals early on, and during later steps more difficult notions are optimized – such as the melody. This finding is in line with existing research which shows that recurrent neural networks are able to define proper music structure while they struggle with the content [Eck and Schmidhuber, 2002].

The results allow us to make another observation: the basic music-specific metrics are useful to an extent. Since the tracked metrics are easy to learn, most experiments achieve similar scores. Surprisingly, standard scores from machine translation like BLEU have proven a more useful heuristic when comparing models.

12.5 Model Must Be Challenged

Students struggle on hard problems more than they do on easier problems. But throwing someone in deep waters teaches them how to really swim. The same seems to apply in machine learning.

One set of three experiments measured how much the trained model struggles to up the performing level of a melody from easy to intermediate, and easy to difficult (the accompaniment in the left hand is ignored). In the latter case, the model seems to learn more.

The trained model always struggles most to go to advanced level, either from beginner or intermediate (judging by the ROUGE-L & BLUE scores). But those are the instances which teach the model the most. The road to enlightenment must indeed be sowed with

Challenge	No. Train. Pairs	ROUGE-L F-1	ROUGE-L Prec.	ROUGE-L Recall	Tonal Certainty	Key Acc.	Mode. Acc.	Note Density	Experiment ID
Beg –> Int	2,174	33%	81%	30%	0.63	28%	55%	1.72	`express_1`
Int –> Adv	1,470	17%	90%	17%	0.67	31%	61%	1.11	`express_2`
Beg –> Adv	1,442	17%	86%	17%	0.67	35%	62%	1.28	`express_3`

Table 9: Expressiveness. Three experiments challenge the trained model to increase the performing difficulty of a melody improving its expressiveness. The left-hand part (the accompaniment) is ignored in those experiments.

dire straits[18]. Shoot for the stars and you might hit the moon. Difficult tasks lead to better metrics in Key Accuracy, Tonal Certainty and Correlation Coefficient (see Table 9).

Generally, in an attempt to write music at an advanced level, given either an easy or intermediate melody, the model achieves high precision and low recall. This means that not much new material is added, but whatever is added is generally correct – and much more correct compared to when the model is not challenged. Observe how precision increases from 81% (Beg –> Int) to 90% and 86%, (Int –> Adv) and (Beg –> Adv).

12.6 Middle Class is the Backbone

Level	No. Train. Pairs	R-L F-1	R-L Prec.	R-L Recall	Corr. Coeff.	Key Acc.	Mode. Acc.	Tempo (bmp)	Tempo Dist.	Duration Dist. (s)	Note Density	Exp. ID
Beginner	2,174	45%	85%	41%	0.61	33%	56%	103	14	16.8	1.25	`acc_1`
Intermediate	2,202	35%	83%	32%	0.77	51%	69%	193	41	9.2	1.12	`acc_2`
Advanced	1,470	12%	80%	11%	0.58	35%	46%	237	28	8.7	0.58	`acc_3`

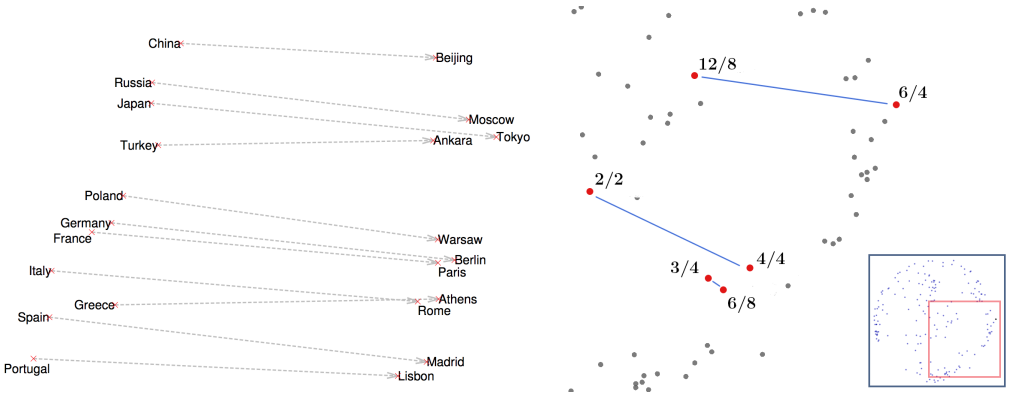
Table 10: Accompaniment. Three experiments challenge the model to add accompaniment to a melody. Both the melody and the accompaniment are at the same level of performing difficulty.

Here is another set of three experiments which supports the previous finding. The current experiment asked the model to provide an accompaniment given a melody. The accompaniment and the melody are of same difficulty level. It turns out that the advanced level is quite a challenge and the scores are below those for the beginner and intermediate levels (see Table 10). The previous sectioned concluded that the advanced level gives the model a healthy dose of challenging exercises. With the results from the current set of experiments, it can be added that while the exercise may be healthy, one would not expect the model to do great on the challenge. However, augmenting advanced-level

[18]No pun intended with Dire Straits, the British rock band of the same name

training examples with data from easier levels may help drive better performance overall (in fact, this hypothesis is confirmed with later experiments). Apparently, the advanced level serves a a sort of *performance booster*, while the grunt work is carried by the "middle class" – the easy and intermediate levels.

12.7 Good Sense for Rhythm



(a) A reproduction of an original illustration from a wildly popular paper by Mikolov et al. with more than 8,000 citations. The picture plots in 2-d word embeddings which reveal implicitly learnt semantic relationships such that vec(*Beijing*) – vec(*China*) + vec(*France*) is closer to vec(*Paris*) than any other vector. Source: [Mikolov et al., 2013]

(b) The current project has unveiled that seq2seq models are able to learn similar implicit relationships between key signatures. The picture above shows that vec(`2/2`) – vec(`4/4`) + vec(`6/4`) is closer to vec(`12/8`) than any other vector, suggesting awareness of doubling note values and relationships between rhythmic patterns. Image by the author.

Figure 28: Two-dimensional PCA projections showing how two trained models, the first from the field of natural language understanding and the second from music informatics, can use similar model architectures to implicitly organize concepts.

One exciting finding is that the model has a firm sense for rhythm which manifests itself in all subjectively examined sequences. It has been practically impossible to find examples when the output meanders without a sense of underlying rhythm. Fig. 28 shows evidence that the trained model implicitly learns relationships between rhythmic patterns and notions of doubling note values.

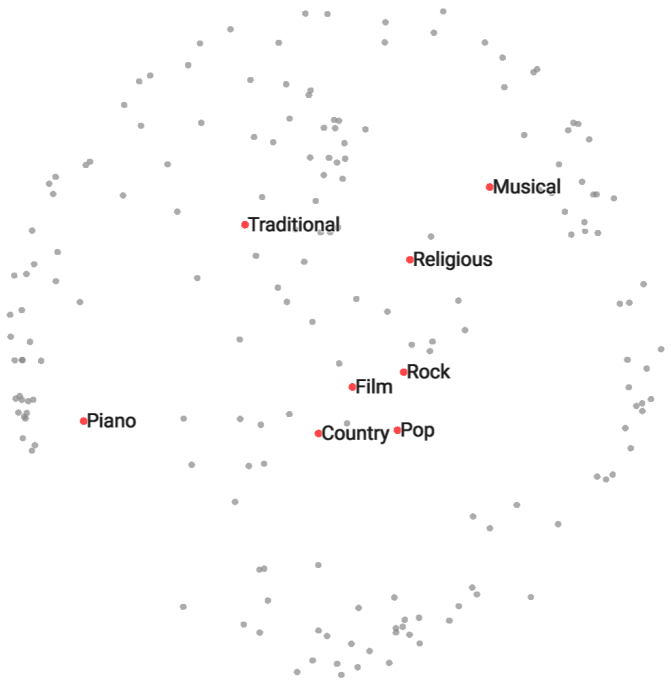
Among the very few examples with imperfect less than perfect rhythm is the output of the holiday song *O Christmas Tree*. The given melody plays beautifully and it is immediately identifiable to untrained ears. However, upon closer listening trained ears can notice that the the outputted melody and harmonized chords sounds a bit in between

a 3/4 and 4/4 meter when the original song is in 3/4. One possible explanation is that the vast majority of the training data is in an either simple or compound duple meter (i.e. 2/4, 4/4, 2/2, 6/4), while simple and compound triples (i.e. 3/4) are the exception from the norm. It is possible that the model has overfitted to duples, or that there are not enough examples to master the triple meter. Taking a second glance at Fig. 28, one can see that the embeddings for meters 3/4 and 4/4 are positioned quite closely in space. Supposedly, with more training examples in triple meters, 3/4 and 4/4 will be pushed away from each with additional training.

12.8 Vastly Different Genres Adequately Transformed

Different genres are processed adequately with some respect to the specifics of the genre. The phenomenon was first observed subjectively by listening to synthesized model output. The discovery prompted a search for a better way to drive the point home, quantitatively. Projecting the embeddings of the trained metadata tokens from experiment `meta_3` provided a useful method to show the learnt relationships, as we can see in Fig. 29.

Interestingly, the embeddings show a high similarity between the genres of pop, rock and country, as well as film music. The fact may not be that surprising though. Our training dataset is arranged for piano, and rock and pop are arguably distinguishable by features which are not present in piano arrangements, such as vocals and representative music instruments. Rock often uses drums, bass and electric guitars, while pop is more often associated with acoustic guitars or electronic sound and dance rhythms. Furthermore, the difference between the two genres arguably has more to do with culture than music. Modern pop music is built around the four-bar meter, and rock is not too different – stressing beats 2 and 4. Rock evolved from country into rock'n'roll which grew into rock, which is typically performed at packed stadiums, while pop has been more of a radio genre. Bearing that in mind, one can see who pop, rock and country, arranged for piano can appear similar. For context, a few examples from the training dataset in different genres are Bryan Adams' *Everything I Do* (rock), ABBA's *Dancing Queen* (pop) and Carrie Underwood's *See You Again* (country).



(a) Visualization of the relationship of tracks of different genres. More similar genres are plotted closer together.

(b) Visualization of the relationship of different song segments. More similar segments are plotted tighter together.

Figure 29: The plots above show plots from a factor analysis dimensionality reduction on the event embeddings from experiment `meta_3`. The shown dimensions capture 73.9% of the variance. The reason why PCA is preferred, as opposed to statistically more advanced techniques like T-SNE, is because T-SNE in practice failed to produce a more visually clear plot. An added benefit of using PCA over T-SNE is the reproducibility of results via through standard analytical operations (T-SNE relies on an optimization engine which depends on the choice of hyper-parameters and converges at different minima on each run).

12.9 Performance Fast Deteriorates with Longer Sequences

Example Length	Avg. Length (secs)	No. Train Pairs	R-L F-1	R-L Prec.	R-L Recall	Corr Coeff.	Key Acc.	Mode. Acc.	Tempo (bmp)	Tempo Dist.	Dur. Dist. (secs)	Note Density	Exp. ID
1 Bar	2 sec	59,349	49%	54%	49%	0.76	33%	62%	208	23	0.1	0.82	`len_1`
2 Bars	4 sec	29,819	47%	61%	44%	0.77	32%	59%	201	36	0.6	0.88	`len_2`
5 Bars	10 sec	12,127	44%	65%	41%	0.78	39%	63%	203	41	1.6	0.99	`len_5`
10 Bars	20 sec	6,225	22%	77%	20%	0.74	46%	64%	196	57	8.4	0.64	`len_10`
Segment *(∼13 bars)*	25 sec	5,086	22%	81%	21%	0.75	49%	68%	211	43	9.5	0.91	`baseline`
20 Bars	40 sec	3,271	19%	83%	18%	0.73	53%	64%	138	72	22.9	0.55	`len_20`
60 Bars	2 min	1,281	8%	83%	8%	0.62	41%	62%	240	49	68.8	0.43	`len_60`
Full Song *(∼100 bars)*	3 min 35 sec	568	3%	92%	3%	0.18	3%	26%	N/A	N/A	147.5	N/A	`len_f`

Table 11: Evolution of model performance with longer sequences.

One of the larger sets of experiments looks at the behavior of metrics as the length of training examples is increased (all reported experiments outside this section use the *segment* as a default chunking unit).

Performance-wise, we can roughly group all experiments into three groups – under 5, between 5 and 20, and more than 20 bars. The shortest examples achieve an F-1 score of 45%, the medium-long hover around 20% and the longest do poorly at 8%. The quality of the predicted arrangements deteriorates incredibly fast for performances longer than 30-40 seconds (about 20 bars). Results are consistent with the existing literature on use of sequence-to-sequence models in general [Cho et al., 2014]. Actually, it is rather surprising that the model is able to handle the long sequences as well as it does. With the chosen `ON`, `OFF` and `SHIFT` encoding, 20-bar sequences are comprised of upwards of 200 tokens – which is way longer than the number of words in natural language sentences. Another reason why decreasing performance is expected is due to the inverse relationship between the number of training examples and length of the examples. When songs are split into fewer chunks, there are many training examples (tens of thousands). As we approach to using the length of whole songs, the training examples are in the hundreds.

Quite naturally, short sequences exhibit much better precision and recall. However, on listening to them, they do not provoke nearly the same sense of trepidation as longer sequences which build a sense of awe that the trained model is able to comprehend such a harmonically, rhythmically and melodically complex composition. Furthermore, the

shortest training examples of length 1 and 2 bars exhibit lower Key Accuracy (about 30%), compared to longer examples which score above 50%.

Another interesting observation is the inverse relationship between precision and recall. Precision rises from 49% to 92%, while recall drops from 49% to 3% as the length of the training examples increases from 1 bar to 120 bars. In other words, the longer the sequence, the more the model "copies" some fragments of the input without adding enough of its own interpretation and expressiveness which characterize higher recall rates. This has been verified subjectively.

Importantly, the model does better when trained on *segments* compared to examples automatically sliced every 10 bars, even though a segment is on average 13 bars. This speaks to the fact that sensible chunking helps the model. Sadly, well-chunked datasets are hard to come by and chunking cannot be done well algorithmically [Maxwell, 2014].

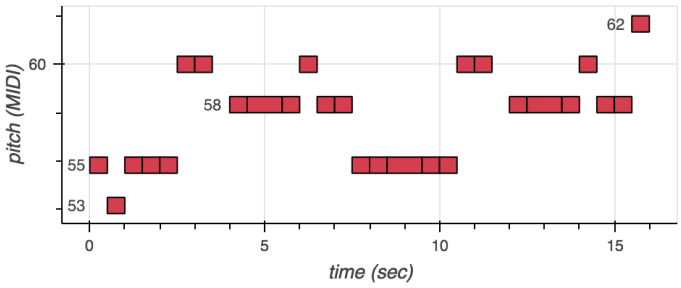
12.10 Time-tested Ideas Still Have Clout

Encoder Architecture	Loss (Eval.)	BLEU	R-L F-1	R-L Prec.	R-L Recall	Corr. Coeff.	Key Acc.	Mode. Acc.	Tempo Dist.	Duration Dist. (s)	Note Density
Bidirectional	1.28	13.92	55%	75%	50%	0.82	45%	67%	32	6.4	0.66
Pyramidal	0.83	14.52	56%	74%	52%	0.82	48%	65%	31	6.2	0.66
Transformer	0.79	7.73	44%	74%	41%	0.79	44%	57%	45	7.8	0.61

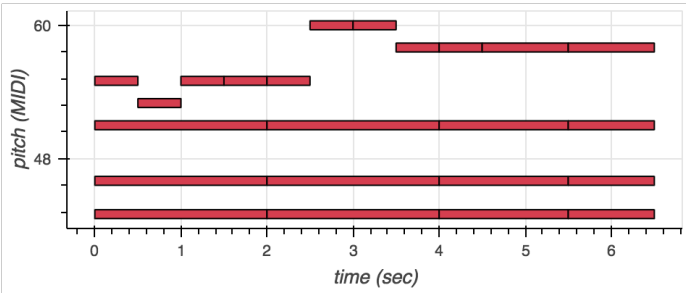
Table 12: Comparison of the performance of three different architectures.

This set of three experiments pits against each other three architectures, each trained on the same large dataset augmented by transposing all songs an octave lower and an octave higher. All three models have a comparable complexity in terms of number of layers and neurons. While it is not immediately obvious from the quantitative metrics which architecture should be preferred, once output is evaluated qualitatively, the picture becomes clearer.

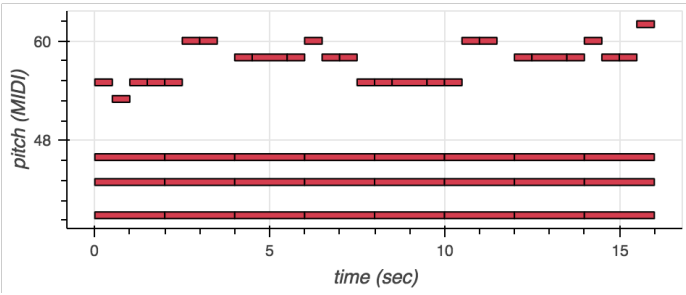
Old ideas in music, specifically Lerdahl and Jackendoff's *Generative Theory of Tonal Music*, apparently still carry clout. The Pyramidal architecture, which imitates L&J's Grouping and Time-Span reduction structures, is able to model melodic input far better than the Tranformer and the Bidirectional architectures. The point is easiest to illustrate with a picture. Fig. 30 compares the Pyramidal and Transformer models head-to-head. Transformer is clearly inferior, producing a rigid melody with little variation and oversimplification of the original melody. On the other hand, the Pyramidal model



(a) A candidate melody for accompaniment, from the evaluation dataset.



(b) Predictions with Transformer model after 15,000 training steps

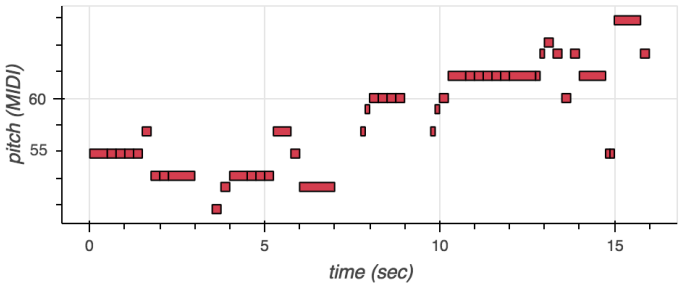


(c) Predictions with Pyramidal model after 4,284 training steps.

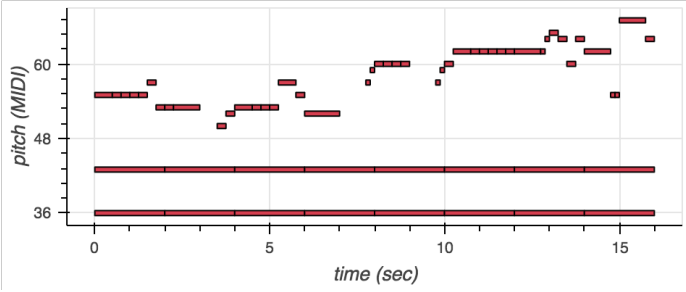
Figure 30: Models which drive lower loss are not always superior. Above, the Transformer model achieves lower loss than the Pyramidal architecture, yet it produces more rigid and spiritless melodies compared to the Pyramidal model. Unfortunately, the accompaniments generated both by Transformer (b) and Pyramidal (c) model make use of repetitive chords.

reproduces the original melody faithfully (even though both models leave more to be desired in terms of chordal harmonization). The example is representative.

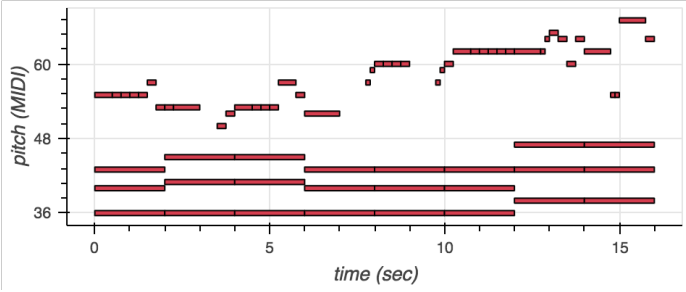
The results from the current set of experiments are also interesting because the superiority of one model over the rest is not immediately clear. The Transformer model achieves the best loss and the worst BLEU score on the evaluation dataset. Given the results from the subjective manual evaluation, the facts are also saying something about the relevance of the BLEU score in music. A gold standard in natural language processing, BLEU apparently carries its wight in the domain of music as well, at least with the data representation chosen for the current project.



(a) Candidate melody for accompaniment (The *Lion King* theme song), from the evaluation dataset.



(b) Predictions using the best model, Pyramidal RNN Encoder, after 13,568 training steps (Eval. Loss = 0.7482).



(c) Predictions using the best model, Pyramidal RNN Encoder, after 79,104 training steps (Eval. Loss = 1.059).

Figure 31: Loss can also be misleading when deciding for how many epochs to train a model. Normally, we want to prevent a model from over-fitting with techniques like regularization and early stopping. Above, an example is shown where over-fitting yields better results. Evaluating from a checkpoint which has overfitted to the training dataset (c) produces a more varied and complex accompaniment to a song in the evaluation dataset, compared to a checkpoint (b) which has lower loss and theoretically should be better.

12.11 Overfitting Helps

Figure 31 shows output from the same model, Pyramidal RNN, after a different number of training steps. What the example reveals may contradict common sense. Surprisingly, overfitting leads to more varied and complex accompaniments. After 79k training steps, the loss on the evaluation set has risen to 1.06 due to overfitting. Yet, the provided harmonization is more varied and more difficult. A comment on an online discussion board by Ian Simon, one of the principal researchers at Google's Magenta project, gives some credibility to these findings. Simon suggest that deliberate overfitting to music data may (or not) yield interesting results.[19]

[19]https://groups.google.com/a/tensorflow.org/forum/#!topic/magenta-discuss/YW361jntg3Q

13 Inside the Black Box

A common criticism of machine learning is that it is a black box which performs transformation researchers cannot explain. The current section aims to lift a bit the veil by visualizing the attention layer of our trained model. The visualization shows what the model is paying attention to while it listens to a melody and how it decide on appropriate chords to a given melody.

Visualizing Attention

Attention visualization is a useful tool which can be used to understand how a trained model is "thinking." Fig. 32 shows the mechanics of how the trained model harmonizes a melody. We see that the model plays 5 times 2 different chord progressions with a strong preference for the C-Major Triad of G-C-E which is used four out of five times. We observe that when the model decides which chord progression to play (events of type `ON`), it spreads its attention over the entire input sequence. Interestingly, when the model is to stop playing a given chord progression (events of type `OFF`), the model focuses its attention on the last three events of the entire input sequence. One reasonable explanation for the observed behavior would be cadence. Tonal music most commonly resolves to the tonal center at the end of a sequence. It is plausible to suppose that the model has noticed the phenomenon and uses the last events from the sequence as a cheating trick.

The attention visualization reveals as well that the trained model concentrates its attention on single events when encoding and decoding a melody, in comparison to chord events. This is not a surprise as the model is expected to preserve and reproduce the inputted melody so the model obviously pays close attention to the priming melody. As a result, there is a close alignment between input and output. The chord progression, on the other hand, is at the model's discretion and the model chooses to spread its attention over the entire available information.

Lastly, the visualization reveals a final phenomenon of interest. Notice the highlighted circle in the far left. It happens to be at the beginning of the decoding process but towards the second half of the encoded sequence. Why does the model jump ahead and focus its attention two thirds into the input? The strongest activation is on an event which indicates "play middle C." The behavior is not clearly understood. Possibly, the model uses that as another trick, this time for recognizing the key of the input melody. In any case, the choice is not arbitrary, as it has been observed in more than one inspected example.

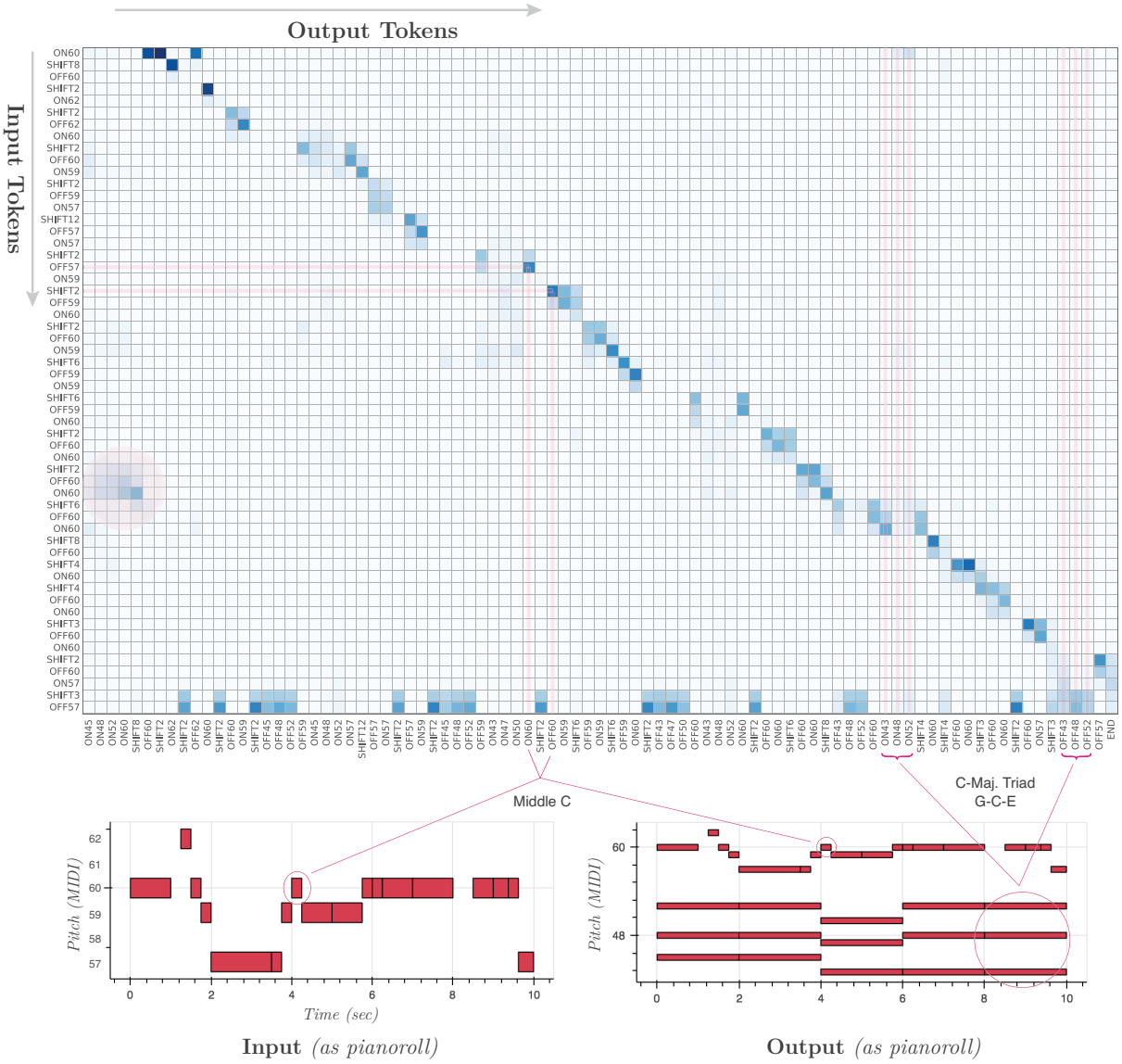


Figure 32: The matrix above shows how much attention the trained model pays to each input token when predicting each output token. The light intensity of the squares corresponds to an activation level determined by the probability output of the softmax function. Darker colors correspond to higher probability. Because the softmax distributes probability mass, the sum of all values in a column adds to 1. On the horizontal axis, left-to-right, we see the output tokens and on the vertical axis, top-to-bottom, we see the input tokens. The two plots below the attention matrix show the same data in a more intuitive form as a pianoroll.

Part VIII
Conclusion

The current research shows the design, implementation and evaluation of a practically working model which generates beautiful accompaniment to a wide variety of input melodies. The research shows the breaking points of RNNs in arranging music with close to a dozen findings backed by quantitative metrics and visualizations of the learned vector embeddings, trained weights and attention activations. The final model, a Pyramidal Encoder RNN architecture with an Attentional Decoder, is successfully transferred for the first time from the field of natural language processing to that of music informatics. A major challenge which the current project deals with is the encoding of the original data in a format suitable for machine learning. The solution is important because existing models for natural language processing are designed to encode one event at a time. This is because language, unlike music, arranges words in a linear sequence along a time axis, while music additionally models many pitches played at once in a single time step.

The success of the current project can be partially attributed to the unique training dataset available for research. As discussed in the beginning, good training data is scarce in the the field of music informatics and a major hindrance to progress. The available dataset, 200 popular songs, each arranged in at least two arrangements of different complexity, offers rarely seen perfect alignment between arrangements down to the single bar. The quality of the data allows its slicing in different subsets which are used to stress test the trained model from various attack points – Can it add accompaniment given a melody? Can it increase the level of a melody's performing complexity? Does it learn to stay in key? Does conditioning on metadata help?

Lastly, the developed infrastructure for subjective interactive and visual evaluation has proven incredibly valuable in interpreting, explaining and developing an intuition around the meaning of dry numerical scores and the complex interplay between dataset variations, architectures and hyper-parameters.

14 Next Steps

The current project paves the roadwork for at least four different continuations, all immediately stemming from the presented work.

First, results point to the fact that it is worth deliberately overfitting without the use of regularization and dropout. In contract to the generation of new melodies, in the

domains of harmonization, arrangement and orchestration, "plagiarizing" chords from one performance to another is not troublesome, even desirable, because there are only that many chords to choose from.

Second, different types of representations should be tested given that data representation directly affects the ability of a model to learn. The current project used a MIDI-inspired data representation which is a *procedural* encoding, i.e. a stream of instructions. Alternatively, one can conceive a more *declarative* representation which is inspired from the way music is traditionally encoded on the staff. Rather than using ON[1-127], OFF[1-127] and SHIFT[16] events, one can use NOTE[1-127][1-16] and SHIFT[1-16] events. The difference is that NOTE has an additional associated value which determines its duration.

Third, the superior performance driven by the use of a Pyramidal RNN Encoder suggests the benefits of using models which explicitly account for the hierarchical nature of music. In future works, one can conceivably design an Encoder which encodes sensible chunks from the underlying musical surface, as opposed to the rigid splitting at fixed intervals which was used in the current project. Of course, this requires first the development and implementation of an algorithm which can chunk musical content. Because chunks are variable-length units, the architecture will have to adjusted as well. Each chunk can be processed by a dedicated RNN which compacts a variable-length chunk into a fixed-sized vector which is then made available to higher levels in the pyramidal structure.

Finally, it will be interesting to take the findings of the current research to *live* music. One application of instantaneous harmonization can be in educational and self-practice settings. A performer can practice with the trained model which would play a *backing track* responding to cues from the performer. Searching YouTube for "backing tracks" returns millions of backing tracks videos sorted (i.e. atmospheric ballad backing track in E, groove blue rock backing track in D, etc) which guitarists play in the background while they play the lead guitar.

15 Final Words

In the final week before the submission of this dissertation, a friend listened to samples generated by the final best model. She heard the input first, the priming melody, and then the output, a rendition of the same melody harmonized with appropriate chords. On hearing a harmonized track, she exclaimed "Aha! This is the Lion King!" The fact is she had not recognized the priming melody as the theme song of the eponymous

Disney animation. However, the output from the model harmonized was immediately recognizable to her. The small anecdote spoke volumes to me and it was one of the biggest compliments which an author can receive after weeks of grueling work focused on getting the details right without loosing the big picture.

Quantitative evaluation has shown that the final best model keeps rhythm and learns meter, including the mathematical relationship between compound time signatures. The model has a notion of different genres and transforms each with some respect to the underlying specifics of the genre. Even though output can sometimes be repetitive and unimaginative, it is almost never nonsensical. The rare exception is input of polyphonic melodies with many short notes and of a very fast difficult rhythm. But even then, when the model struggles to reproduces the given melody faithfully, it applies a sensible transformation in the same key and time signature as the original.

The current research also goes on to show that the challenge for recurrent-neural-network-based models is not necessarily the length of the input but rather the expectation that the network crafts semantically meaningful content. What is difficult for RNNs is to create *original content*, an area which remains still predominantly in the domain of human creativity. That said, what RNNs excel at is learning and applying the fundamentals of theoretical knowledge, including music theory – i.e. learning the scale, finding the tonal center of a composition, determining its rhythm, beat and transforming an input sensibly with respect to its defining characteristics.

The findings support the motivation of the current work to encourage focus on models which allow for human-computer interaction as opposed to the design of one-click deep neural generators of tunes deprived of a creative spark. There is a lot of potential that can be harvested. Especially if appropriate datasets are created, a number of solutions from the fields of natural language processing can be repurposed to advance the state of music problems, some as ambitious as automated orchestration.

Lastly, using the most fashionable architecture does not guarantee the best results, as the battle between the cutting-edge Transformer model and the slightly older Pyramidal RNN has shown. What matters however is taking an approach with respect to the properties of the underlying data. A pyramidal encoding architecture matches better the hierarchical nature of music formalized by Lerdahl and Jackendoff's 1983 *Generative Theory of Tonal Music*. Drawing a bridge between older ideas in music to the latest fashion in deep learning can be powerful, and this is the last point which this report will make. Progress happens when new ideas are combined mindfully with respect to classic theory which has stood the test of time.

The table below summarizes and compares the state-of-the-art architectures for algorithmic music composition. The focus is on models which go beyond click-and-generate by allowing *steering* with a user-provided input.
The table is by the author. Previously published with the author's Informatics Research Review project at The University of Edinburgh in January 2018.

Short name	**Year**	**Architecture**	**Training Corpus**	**Representation**	**Domain knowledge**	**Steerable**	**Noteworthiness**
Modelling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation [Boulanger-Lewandowski et al., 2012]							
RNN-RBM	2012	x1 RNN which determines the bias of a RBM at each step [Gibbs Sampling]	67 hours of polyphonic music (piano, folk, orchestral, chorales)	Custom (analogous to ABC Text Format)	No	No	- Simple yet powerful: learns transition probabilities between notes & produces pleasantly sounding melodies
Imposing High Structure in Polyphonic Music Generation using Convolutional Restricted Boltzmann Machines and Constraints [Lattner et al., 2016]							
C-RBM	2016	C-RBM with 512 input nodes, 2048 hidden units – convolution on time only, not pitch [Gibbs Sampling]	Rather small: 3 of Mozart's sonatas (No. 1 C major, No. 2 F major, No. 3 B major) = 15,144 time-steps	Piano roll with 512 time steps & 64 pitches	No	Yes	- With a cost function, user can control 1) key, 2) meter and, 3) the global structure's similarity to a template (i.e. an 'inspiration' like a melody or a motif)
A Steerable Model for Bach Chorales Generation [Hadjeres and Pachet, 2016]							
DeepBach	2016	x2 LSTM RNN + x2 Feedforward NNs [pseudo-Gibbs Sampling]	2503 J. S. Bach chorales harmonized in 4 voices	MIDI pitches + additional modelling	No	Yes	- Users can impose unary constraints - Convincing results
Tuning Recurrent Neural Networks with Reinforcement Learning [Jaques et al., 2016]							
RL-Tuner	2016	RNN with one layer of 100 neurons, tuned by a non-differentiable function through RL.	30,000 monophonic songs	MIDI	Yes	No	- Good global structure made of phrases; doesn't wander - Can be explicitly given domain knowledge
A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation [Yang et al., 2017]							
MidiNet	2017	Generative Adversarial Network comprised of a Generator CNN, Discriminator CNN, Conditioner CNN	1,022 pop melodies from the TheoryTab dataset. 50,496 bars of melodies after data augmentation through transposition	Piano roll, derived from MIDI	Yes	Yes	- Can generate from scratch or exploit prior knowledge (i.e. priming with a melody or a user-defined chord progression) – State-of-the-art GAN architecture, comparable performance to Google's *MelodyRNN* which is one of the state-of-the-art models overall

References

[Allan and Williams, 2005] Allan, M. and Williams, C. (2005). Harmonising chorales by probabilistic inference. In *Advances in neural information processing systems*, pages 25–32.

[Alpern, 1995] Alpern, A. (1995). Techniques for algorithmic composition of music. *On the web: http://hamp.hampshire.edu/adaF92/algocomp/algocomp*, 95:120.

[Antonopoulos, 2011] Antonopoulos, A. (2011). Pithoprakta: The historical measures 52-59 new evidence in glissando speed formalization theory & theoretical applications. In *Proceedings of the Xenakis International Symposium, London*, pages 1–3.

[Ariza, 2009] Ariza, C. (2009). The interrogator as critic: The turing test and the evaluation of generative music systems. *Computer Music Journal*, 33(2):48–70.

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[Barnes, 2014] Barnes, T. (2014). Why not being able to read music means noth- ing about your musical ability.

[Boulanger-Lewandowski et al., 2012] Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*.

[Bretan et al., 2016] Bretan, M., Weinberg, G., and Heck, L. (2016). A unit selection methodology for music generation using deep neural networks. *arXiv preprint arXiv:1612.03789*.

[Brownlee, 2017] Brownlee, J. (2017). Gentle introduction to the adam optimization algorithm for deep learning.

[Callison-Burch et al., 2006] Callison-Burch, C., Osborne, M., and Koehn, P. (2006). Re-evaluation the role of bleu in machine translation research. In *11th Conference of the European Chapter of the Association for Computational Linguistics*.

[Chan et al., 2016] Chan, W., Jaitly, N., Le, Q., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4960–4964. IEEE.

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

[Coughlin, 2003] Coughlin, D. (2003). Correlating automated and human assessments of machine translation quality. In *Proceedings of MT summit IX*, pages 63–70.

[Cross, 1998] Cross, I. (1998). Music analysis and music perception. *Music Analysis*, 17(1):3–20.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE.

[Dixon, 2001] Dixon, S. (2001). Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research*, 30(1):39–58.

[Eck and Lapalme, 2008] Eck, D. and Lapalme, J. (2008). Learning musical structure directly from sequences of music. *University of Montreal, Department of Computer Science, CP*, 6128.

[Eck and Schmidhuber, 2002] Eck, D. and Schmidhuber, J. (2002). Learning the long-term structure of the blues. *Artificial Neural Networks—ICANN 2002*, pages 796–796.

[Eerola et al., 2001] Eerola, T., Jäärvinen, T., Louhivuori, J., and Toiviainen, P. (2001). Statistical features and perceived similarity of folk melodies. *Music Perception: An Interdisciplinary Journal*, 18(3):275–296.

[El Hihi and Bengio, 1996] El Hihi, S. and Bengio, Y. (1996). Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems*, pages 493–499.

[Engel et al., 2017] Engel, J., Resnick, C., Roberts, A., Dieleman, S., Eck, D., Simonyan, K., and Norouzi, M. (2017). Neural audio synthesis of musical notes with wavenet autoencoders. *arXiv preprint arXiv:1704.01279*.

[Fabius and van Amersfoort, 2014] Fabius, O. and van Amersfoort, J. R. (2014). Variational recurrent auto-encoders. *arXiv preprint arXiv:1412.6581*.

[Fred Lerdahl, 1983] Fred Lerdahl, R. J. (1983). Generative theory of tonal music.

[Genthial, 2017] Genthial, G. (2017). Seq2seq with attention and beam search.

[Gibson, 2011] Gibson, B. (2011). *The Instrumental Music of Iannis Xenakis. Theory, Practice, Self-Borrowing*. Pendragon Press.

[Graves et al., 2013a] Graves, A., Jaitly, N., and Mohamed, A.-r. (2013a). Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE.

[Graves et al., 2013b] Graves, A., Mohamed, A.-r., and Hinton, G. (2013b). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE.

[Hadjeres and Pachet, 2016] Hadjeres, G. and Pachet, F. (2016). Deepbach: a steerable model for bach chorales generation. *arXiv preprint arXiv:1612.01010*.

[Handel, 1993] Handel, S. (1993). *Listening: An introduction to the perception of auditory events.* The MIT Press.

[Harris and Harris, 2010] Harris, D. and Harris, S. (2010). *Digital design and computer architecture*. Morgan Kaufmann.

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

[Jaques et al., 2016] Jaques, N., Gu, S., Turner, R. E., and Eck, D. (2016). Tuning recurrent neural networks with reinforcement learning. *arXiv preprint arXiv:1611.02796*.

[Jean et al., 2014] Jean, S., Cho, K., Memisevic, R., and Bengio, Y. (2014). On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*.

[Kalchbrenner and Blunsom, 2013] Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709.

[Karpathy, 2015] Karpathy, A. (2015). The unreasonable effectiveness of recurrent neural networks. *Andrej Karpathy blog*.

[Karpathy and Fei-Fei, 2015] Karpathy, A. and Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kirchmeyer, 1968] Kirchmeyer, H. (1968). On the historical constitution of a rationalistic music. *Die Reihe*, 8.

[Klein et al., 2017] Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. M. (2017). Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*.

[Koehn et al., 2003] Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics.

[Lattner et al., 2016] Lattner, S., Grachten, M., and Widmer, G. (2016). Imposing higher-level structure in polyphonic music generation using convolutional restricted boltzmann machines and constraints. *arXiv preprint arXiv:1612.04742*.

[LeCun and Cortes, 2010] LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

[Lerdahl and Jackendoff, 1985] Lerdahl, F. and Jackendoff, R. S. (1985). *A generative theory of tonal music*. MIT press.

[Lin, 2004] Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*.

[Luong et al., 2014] Luong, M.-T., Sutskever, I., Le, Q. V., Vinyals, O., and Zaremba, W. (2014). Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*.

[Luque, 2009] Luque, S. (2009). The stochastic synthesis of iannis xenakis. *Leonardo Music Journal*, pages 77–84.

[Makris et al., 2017] Makris, D., Kaliakatsos-Papakostas, M., Karydis, I., and Kermanidis, K. L. (2017). Combining lstm and feed forward neural networks for conditional rhythm composition. In *International Conference on Engineering Applications of Neural Networks*, pages 570–582. Springer.

[Marcus et al., 1993] Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

[Markoff, 2015] Markoff, J. (2015). A learning advance in artificial intelligence rivals human abilities. *New York Times*, 10.

[Maxwell, 2014] Maxwell, J. B. (2014). *Generative Music, Cognitive Modelling, and Computer-Assisted Composition in MusiCog and ManuScore*. PhD thesis, Simon Fraser University.

[Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[Mistler, 2017] Mistler, E. (2017). Generating guitar tablatures with neural networks. *Master of Science Dissertation, The University of Edinburgh*.

[Mogren, 2016] Mogren, O. (2016). C-rnn-gan: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*.

[Obacom, 2015] Obacom (2015). 10 legendary musicians who never learned how to read music.

[Pachet and Aucouturier, 2004] Pachet, F. and Aucouturier, J.-J. (2004). Improving timbre similarity: How high is the sky. *Journal of negative results in speech and audio sciences*, 1(1):1–13.

[Papadopoulos et al., 2016] Papadopoulos, A., Roy, P., and Pachet, F. (2016). Assisted lead sheet composition using flowcomposer. In *International Conference on Principles and Practice of Constraint Programming*, pages 769–785. Springer.

[Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

[Raphael and Stoddard, 2004] Raphael, C. and Stoddard, J. (2004). Functional harmonic analysis using probabilistic models. *Computer Music Journal*, 28(3):45–52.

[Rush, 2018] Rush, A. (2018). The annotated transformer. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 52–60.

[Schmeling, 2011] Schmeling, P. (2011). *Berklee Music Theory*. Berklee Press.

[Schuster and Paliwal, 1997] Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

[Sennrich et al., 2017] Sennrich, R., Firat, O., Cho, K., Birch, A., Haddow, B., Hitschler, J., Junczys-Dowmunt, M., Läubli, S., Barone, A. V. M., Mokry, J., et al. (2017). Nematus: a toolkit for neural machine translation. *arXiv preprint arXiv:1703.04357*.

[Sturm et al., 2016] Sturm, B. L., Santos, J. F., Ben-Tal, O., and Korshunova, I. (2016). Music transcription modelling and composition using deep learning. *arXiv preprint arXiv:1604.08723*.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[Tikhonov and Yamshchikov, 2017] Tikhonov, A. and Yamshchikov, I. P. (2017). Music generation with variational recurrent autoencoder supported by history. *arXiv preprint arXiv:1705.05458*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

[Vinyals et al., 2015a] Vinyals, O., Kaiser, Ł., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. (2015a). Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781.

[Vinyals and Le, 2015] Vinyals, O. and Le, Q. (2015). A neural conversational model. *arXiv preprint arXiv:1506.05869*.

[Vinyals et al., 2015b] Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015b). Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164.

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

[Xu et al., 2015] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057.

[Yang et al., 2017] Yang, L.-C., Chou, S.-Y., and Yang, Y.-H. (2017). Midinet: A convolutional generative adversarial network for symbolic-domain music generation. In *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR'2017), Suzhou, China*.