A Physical Intelligent Instrument using Recurrent Neural Networks

Torgrim Rudland Næss

Thesis submitted for the degree of
Master in Robotics and Intelligent Systems
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2019

A Physical Intelligent Instrument using Recurrent Neural Networks

Torgrim Rudland Næss

A Physical Intelligent Instrument using Recurrent Neural Networks

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Composing and playing music generally requires knowledge of music theory and exercise in instrument training. While traditional musical instruments often require years of arduous practice to master, intelligent musical systems can provide an easier introduction into music creation for novice users. This thesis describes the design and implementation of a novel intelligent instrument for interactive generation of music with recurrent neural networks, allowing users with little to no musical experience to explore musical ideas.

Even though using neural networks for music composition is not a new concept, most previous work in this field does not ordinarily support user interaction, and is often dependent on general-purpose computers or expensive setups to implement. The proposed instrument is self-contained, running an RNN-based generative music model on a Raspberry Pi single-board computer for continuous generation of monophonic melodies that are sonified using a built-in speaker. It supports real-time interaction where the user can modify the generated music by adjusting a set of high-level parameters: sampling temperature (diversity), tempo, volume, instrument sound selection, and generative model selection.

A user study with twelve participants was conducted to see the impact the different high-level parameter controls can have on a participant's perceived feeling of control over the musical output from the instrument, and to evaluate the generative models trained on different datasets in terms of musical quality. The numerical ratings and open-ended answers were analyzed both quantitatively and qualitatively. The results show that the perceived feeling of control over the music was quite high, and the high-level parameter controls allowed participants to creatively engage with the instrument in the music-making process.

Contents

List of Figures

List of Tables

Abbreviations

ADC	Analog-to-Digital Converter
ANN	Artificial Neural Network
BPM	Beats Per Minute
CNN	Convolutional Neural Network
DAW	Digital Audio Workstation
DMI	Digital Musical Instrument
LSTM	Long Short-Term Memory
MIDI	Musical Instrument Digital Interface
NIME	New Interfaces for Musical Expression
RNN	Recurrent Neural Network
SBC	Single-Board Computer

Acknowledgements

I want to thank my supervisor, Charles Martin, for his ideas, guidance, and motivational speeches that kept me going during my work with this thesis. I also want to thank my parents for financial and emotional support, and my roommates, Magna Karina and Thomas, for being there and cheering me on when I had self-doubts. Most of all, I want to thank my fiancée, Aistė, for invaluable help and emotional support, and for keeping me sane during the writing process. I could not have done this without you!

Chapter 1

Introduction

1.1 Motivation

Music has always had an important role in our society. Singing and dancing are often central elements of parties and gatherings, and music provides us with an excellent tool for the expression of feelings. While composing and playing instruments have generally required knowledge of music theory and instrument training, recent advances in computer technology have opened up a world of possibilities for the creation of new kinds of music and musical instruments [1]. Digital Audio Workstations (DAWs) allow anyone with access to basic computer equipment to record and make music of their own. The increased processing power of single-board computers (SBCs) also makes it possible to create cheap, portable embedded devices that can be used for musical performances.

While regular instruments are often expensive and time consuming to master, inexpensive computer hardware and intelligent algorithms give the potential to provide an easier introduction to music for novice users. The use of machine learning algorithms for musical applications, such as music generation [2] and gestural interpretation [3], has generated wide interest among researchers in recent years. With the availability of existing machine learning models from resources such as Magenta [4], it is now easier than ever to delve into the field of computer-generated music.

This thesis presents a self-contained interactive instrument that uses recurrent neural networks to generate continuous monophonic music in real-time, which the user can manipulate using different parameters. This contrasts with most of the previous work in the field of music generation, which tends to be focused on offline creation of musical scores [5], rather than interactive music generation. An overview of the system is shown in Figure 1.1. It has a generative model running on a single-board computer, and an attached speaker to make it self-contained. A set of controls allow for adjustment of high-level parameters to change the musical output.

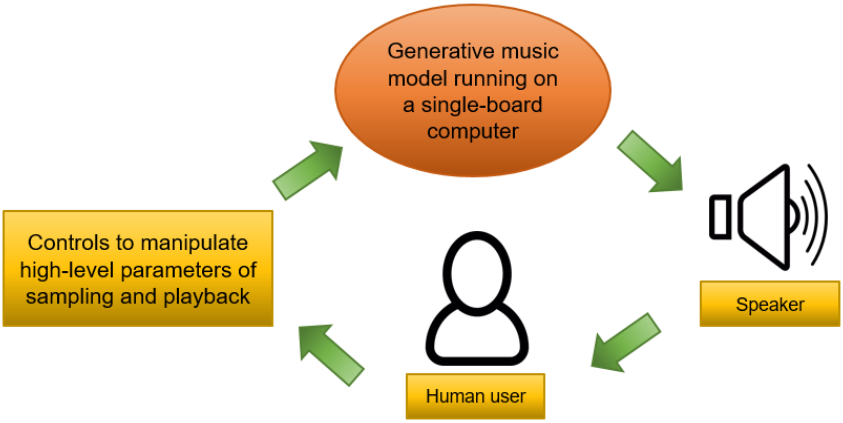


Figure 1.1: System overview of the intelligent musical instrument. A single-board computer (SBC) runs a music generation model to automatically generate and play melodies while the user can interact with controls to change high-level parameters of sampling and playback.

1.2 Goals

The purpose of this thesis is two-fold: to explore the potential of intelligent musical instruments on embedded devices using novel machine learning algorithms to generate music, and to build an interactive environment for users to explore musical ideas. There are two main goals:

- Explore the potential of machine learning algorithms to generate music and investigate how high-level parameters can be used to shape the musical output.
- Design and build a self-contained intelligent musical instrument on an embedded device.

1.3 Research Methods

The work done in this thesis is evaluated using two different approaches. The first is a subjective evaluation of the generative models and how the instrument meets a set of pre-determined design requirements. This is performed by the author. The second approach is a user study to get feedback from multiple performers. In the study, participants explore the instrument and provide feedback through a set of questionnaires with Likert scale evaluations and open-ended queries. The answers are analyzed both quantitatively and qualitatively using statistics and thematic analysis.

1.4 Thesis Outline

This thesis consists of five chapters: introduction, background, design and implementation, testing and evaluation, and conclusion. Chapter 2 is an overview of related work and machine learning concepts relevant to the development of the instrument. Chapter 3 includes a description of the design and implementation process, including physical design and the machine learning approach used for generating melodies. Two instrument prototypes are presented here: an earlier version with only a few high-level parameter controls, and an improved version with more functionality for music creation. The former is featured in an article the author presented at the New Interfaces for Musical Expression (NIME) [6] conference 2019 in Porto Alegre (See Appendix A). Chapter 4 covers the methods for evaluation of the instrument, including a user study for quantitative and qualitative evaluation. Results of the evaluation and discussion of their implications are also included in this chapter. Finally, Chapter 5 is comprised of the conclusions of the thesis in addition to ideas for future work and design improvements.

Chapter 2

Background

This chapter provides an overview of previous work and concepts relevant to the development of the instrument presented in this thesis. An introduction to New Interfaces for Musical Expression (NIMEs) and embedded music systems is presented first, followed by a review of machine learning algorithms for music applications, interactive music systems using machine learning, and finally some background theory of artificial neural networks (ANNs) relevant for music systems.

2.1 New Interfaces for Musical Expression

Inexpensive computing hardware and modern software allowing to synthesize and manipulate real-time sound have made computer instruments common in musical performances [7]. Development of such systems has piqued the interest of many researchers worldwide, who present their work at the New Interfaces for Musical Expression (NIME) conference. The conference started as a workshop at the Conference on Human Factors in Computing Systems (CHI) in 2001, and has since become an annual international conference. NIME research includes novel music systems such as gestural interfaces [8], and musical games [9] and installations [10].

Some NIMEs use algorithmic composition software to allow people with limited musical background to create and play music without spending years mastering a traditional instrument. An example of such a system is a public display created by Gilbert Beyer and Max Meier [11]. The user can only manipulate a few parameters, and the software makes sure that the generated music is perceived as harmonic and resembles known musical themes. The parameters are extracted from the user's movements with optical tracking. Music is generated with two parameters: pitch (high/low) and energy (fast/slow). The only musical skill needed is the ability to make rhythmic movements. These kinds of easy-to-use music systems have also been tested in music therapy, allowing patients to express themselves through music even if intellectual learning disabilities or lack of fine-motor skills would otherwise prevent them from mastering traditional musical instruments [12], [13].

The instrument presented in this thesis can be considered a NIME, since it uses machine learning for automatic melody generation in an interactive manner, and the use of machine learning algorithms for musical applications is becoming a common topic at the conference.

2.2 Music on Embedded Devices

Desktop computers and laptops are a well-established central component of digital musical instruments, but the use of single-board computers and embedded systems is becoming increasingly common. With increasing computing power and availability of single-board computers and microcontrollers, new platforms for creating digital musical instruments appear. These platforms make it easy to prototype embedded instruments for performances and installations [10].

One such platform is Bela [14], which is an expansion board for the BeagleBone Black embedded computer [15]. The hardware of the Bela platform provides powerful real-time processing with low latency. Bela meets all requirements of a self-contained device, providing a high number of I/Os, power output for speakers and a fair amount of processing power, making it an excellent choice to use for embedded digital musical instruments [16]. All hardware and software is open source, and there is also a community where musicians and developers can share ideas and inspiration.

Sverm-resonans (Figure 2.1) [10] is an example of a music installation using Bela. Six acoustic guitars are each equipped with a Bela, IR distance sensor, an actuator, and a battery pack. There are no external speakers attached; the actuator vibrates the body of the guitar to generate sound. An interesting aspect of this system is that the instruments are 'inverse', where the performance is controlled by the absence of motion. The presence of a person is detected by infrared sensors, and the collected data is used to control the generated sound. The inverse relationship between detected motion and amplitude of the sound means that more sound is generated while the person stands still for extended periods of time.

Figure 2.1: *Sverm-resonans*, a musical installation using Bela. Sound is generated by vibrations in the guitar body caused by an attached actuator and it is controlled by the absence of motion from the performer [17].

Satellite CCRMA [18] is another platform designed for musical interaction and sound synthesis. It is built to enable easy design and creation of NIMEs and sound installations, and is completely sufficient to synthesize and generate sound on its own. The platform is based on a Beagle Board embedded computer running Linux, with a microcontroller and breadboard allowing for simple expansion and reconfiguration with new hardware. These properties make it ideal for prototyping new kinds of instruments and it has been used for teaching courses and workshops. Support for Raspberry Pi has also been added to reduce the cost of the kit [19].

Self-contained, or embedded, instrument designs come with several advantages over the use of, for example, a laptop and simple microcontroller-based interface. The increased processing power of single-board computers allows more computationally intensive tasks to be performed natively than on a microcontroller, eliminating the need for external computers. Removing general-purpose computers that are not dedicated to the instrument prototype from the system can also increase longevity, as changes in other software might affect its functionality [20]. In addition, reducing the required amount of wires means that prototypes can operate longer with reduced maintenance requirements [21]. Other advantages include stability and portability [10], which suggests that they can be useful to artists who apply them within instrumental setups during live performances, or in their studios.

2.2.1 Single-Board Computers

Single-board computers (SBCs) have all the features required of a functional computer, such as microprocessors, memory, and inputs/outputs, but they are implemented on a single circuit board. Low power consumption and versatility along with low cost, make them ideal for prototyping mobile and portable systems [22]. It is common for SBCs to run open-source operating systems like Linux, and there are often large communities and forums where people can discuss projects and share source codes.

There are many different SBCs on the market. One of the most widely used is the Raspberry Pi (Figure 2.2) [23], which is a collection of credit-card sized SBCs developed by the Raspberry Pi Foundation. Other examples include Asus Tinkerboard [24], which is more powerful than a Raspberry Pi, but also more expensive, and BeagleBone Black, which is popular in embedded music applications because of the Bela expansion. Some SBCs are better suited for machine learning purposes because they are equipped with powerful GPUs, more RAM and faster CPUs, such as the Nvidia Jetson series [25]. The latter, however, is much more expensive.

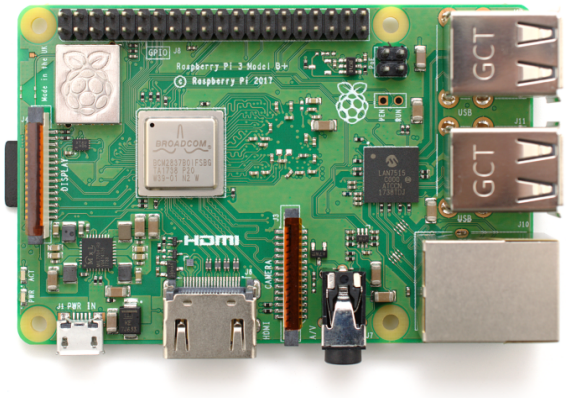


Figure 2.2: A Raspberry Pi 3 Model B+ single-board computer. Photo by Gareth Halfacree, licenced under CC BY-SA 2.0.

2.3 Machine Learning in Music Technology

The use of machine learning algorithms for musical applications has generated wide interest among researchers in recent years. Artificial neural networks (ANNs) can, for example, be applied to the synthesis of musical audio waveform data [26], or translation of music across musical instruments, genres, and styles [27]. Because neural networks can be trained to produce data learned from real-world examples, they are good candidates for the creation of musical scores or performances without the need to manually program the rules of music theory.

Since music can be seen as a sequence of notes, the network must be able to predict notes based on both the current input and on what has been played earlier for the generated output to have musical coherency. Architectures such as Recurrent Neural Networks (RNNs) are designed for the purpose of working with data sequences. Mozer's *CONCERT* network [28], which generates music on a note-by-note basis, is one example of an RNN managing this task. Despite the RNNs theoretical ability to 'remember' previously played notes, the music generated by CONCERT lacked structure. While a standard RNN is, in theory, able to capture long-term dependencies, it is in practice limited by vanishing gradients. Long Short-Term Memory (LSTM), introduced by Hochreiter in 1997 [29], is an RNN architecture designed to deal with the vanishing gradient problem. Both standard RNNs and LSTMs will be discussed further in section 2.5.1.

In 2002, Eck et al. [30] demonstrated that LSTM recurrent neural networks could be used to successfully compose well-structured music, where earlier attempts with standard RNNs tended to lack coherence. Their system learned to compose blues music with timing and structure appropriate for the style. In recent years, LSTM RNNs have been used in a variety of applications, such as generating monophonic melodies resembling specific musical styles [31], [32], automatic generation of chord progressions and harmonies to a melody [33], [34], and creating polyphonic music with expressive timing and dynamics [5].

Some other systems for music generation include Markov models, which are used to model temporal sequential processes. Markov models conceptualize randomly changing systems as a set of distinct states and transitions between those states, and model the system by calculating the probabilities of transitions from one state to another [35, pp. 213–226]. Using Markov models to generate musical structure was first applied around 1950 [36, p. 71] and has since been a common method in musical applications, such as computer-aided composition [2] and harmonization [37]. This method is used in SongSmith [38], a system that produces accompaniment for vocal melodies by automatically generating appropriate chords. The user only needs to sing into a microphone, and the system chooses chords accordingly allowing both expert and novice musicians to experiment with different musical genres and chord progressions.

One disadvantage of using Markov models for generating coherent musical output is that they tend to 'forget' the earliest states and are therefore unable to capture the complete structure of a musical piece, which often entails distant data dependencies [32]. Learning such distant dependencies also requires large transition matrices. In addition, Markov models can only reproduce learned examples [39]. RNNs, on the other hand, are able to make 'fuzzy' predictions, i.e., instead of attempting to match training examples precisely, they interpolate between them [40].

2.3.1 Magenta

Magenta is a research project under Google's direction, initiated by members of the Google Brain team, with the aim to examine how machine learning can be used in the process of creating art, including music [4]. Their research involves both development of new intelligent algorithms for the generation of creative material like music, images, and drawings, and creation of tools that artists and musicians can use to aid them in their craft. Another important aspect of Magenta is the community of artists, coders, and machine learning experts who use the open-source models developed by the Magenta team and released on GitHub. The Magenta team recently released a large dataset dataset comprising over 200 hours of piano concert preformances by skilled pianists [41] as well, which can be used to train machine learning models.

2.4 Musical Interaction with Machine Learning

Some musical systems incorporate machine learning and artificial intelligence to aid musicians in their musical performances, or to simplify the music-making process for novice musicians. There is now potential to embed neural networks within smart musical instruments [42], or to create self-contained ANN music generators that could be used on stage or in the studio. The Magenta team has created physical instruments such as *Piano Genie* (Figure 2.3) [43], which is a recurrent neural network-based system that allows users to improvise on the piano by mapping 88-key piano sequences onto 8-button sequences, and *NSynth Super* [44], a synthesizer using deep neural networks to synthesize sound. Neural networks can also be used to aid musicians in live performances, such as intelligent drum machines that are able to generate variants of rhythmic patterns provided as input by the musician [45].

Figure 2.3: *Piano Genie*, a recurrent neural network-based system that allows users to improvise on the piano [43].

Other musical systems use the input provided by a musician to continue the performance. For example, *Continuator* [46] is a digital musical instrument (DMI) that uses Markov models to learn the stylistic patterns of an individual's performance and uses them to continue playing once the performer stops. Besides Markov models, Deep RNN models have also proven to be able to tackle this task, as demonstrated in *AI Duet* [47]. It is a system that allows users to play a duet with the computer by using a keyboard (whether computer or MIDI).

Another possible application of machine learning is ensemble interactions to emulate the experience of collaborating with other musicians. Martin et al. demonstrate how LSTM recurrent neural networks can be used to create free-form ensemble improvisations using touchscreen apps [48]. Similarly, *Robojam* (Figure 2.4) [49] uses an interface on a touchscreen to allow users to compose brief music samples that can be shared with other users. If desired, Robojam can respond to the user's input by providing musical feedback based on the user's improvisation.

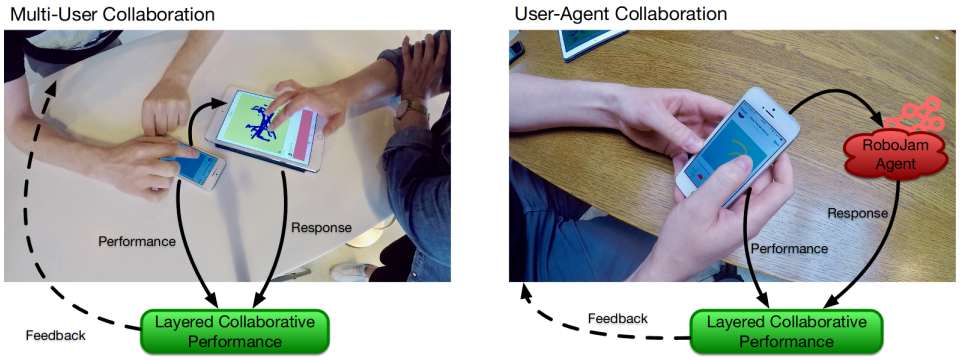


Figure 2.4: *Robojam* is a touch-screen music app where the system can generate musical responses to the performer's compositions [49].

2.5 Artificial Neural Networks for Music Generation

The basic building blocks of ANNs are artificial neurons that take multiple input values, multiply them by respective weights, and produce an output value. The input values can represent anything from pixels in an image to note values. Large numbers of these neurons are interconnected and arranged in a series of layers to form a network that can be trained for different applications such as pattern recognition and image classification. During training, the network is exposed to large numbers of examples with input values and expected output values. Based on the errors encountered in processing these examples, the network will adjust its weights to adapt to the training data.

2.5.1 Recurrent Neural Networks

A recurrent neural network (RNN) is a neural network designed to process a sequence of values [50, p. 196]. It has a hidden state that depends not only on the input at the current time step but also on the state from the previous time step, allowing the network to remember things it saw in the past.

A way to view a recurrent neural network is to unfold it across time (Figure 2.5). At each point in the sequence, we feed in the next input value $\mathbf{x}^{(t)}$and the previous hidden state $\mathbf{h}^{(t-1)}$ and compute the next state $\mathbf{h}^{(t)}$. The state at time t can contain information from all past time steps, which makes an RNN able to learn temporal structures [30]. By sharing parameters across the sequence, it is possible to generalize across sequences of different lengths as opposed to only those seen during training. Parameter sharing is especially important if a sequence can contain the same piece of information at multiple positions, for example, in language modeling where two sentences might have the same meaning even if the words are arranged in a different order.

Connections in the network are parametrized by weight matrices $\mathbf{U}$ (input to hidden connections), $\mathbf{W}$ (hidden-to-hidden recurrent connections)

and **V** (hidden-to-output connections). **b** and **c** are bias vectors. With an initial state set to $\mathbf{h}^{(0)}$, equations 2.1-2.4 [51, p. 374] are applied for every time step t during forward propagation of the network.

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \tag{2.1}$$

$$\boldsymbol{h}^{(t)} = g_h(\boldsymbol{a}^{(t)}) \tag{2.2}$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \tag{2.3}$$

$$\hat{\boldsymbol{y}}^{(t)} = g_y(\boldsymbol{o}^{(t)}) \tag{2.4}$$

A common algorithm for efficient calculation of gradients for RNNs is backpropagation through time (BPTT) [52]. This works much in the same manner as normal backpropagation for a feedforward network, except that the error is calculated for each time step and then accumulated to a combined value. The total gradient then becomes the sum of the gradients at each time step. One weakness with a standard RNN is that long-term dependencies become difficult to learn due to vanishing gradients. Because parameters are shared across time steps, the chain rule products will become very long as the number of steps increases, causing the gradients to either decrease or increase exponentially [30].

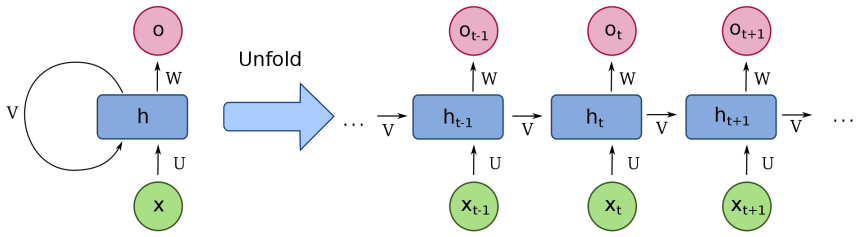


Figure 2.5: RNN unfolded across time. Illustration by François Deloche, licenced under CC BY-SA 4.0

Long Short-Term Memory

Long short-term memory (LSTM) is a commonly used variant of RNNs that is able to capture much longer sequences than a standard RNN, which has a tendency to forget information when the sequences become longer [50, pp. 202-204]. The LSTM achieves this by having an additional internal state called a 'cell state', which can be visualized as a conveyor belt (the horizontal line at the top of the diagram in Figure 2.6) where information is allowed to flow unchanged. The LSTM can also remove or add information to the cell state by using multiplicative gates. There are three gates: an input gate, an output gate, and a 'forget gate'. The input gate controls

the information added to the cell state, making sure that irrelevant inputs do not pass through. Similarly, the output gate makes sure that irrelevant content is not passed along to the next time step. The forget gate allows LSTM cells to reset when the content is no longer needed. The output of these gates are vectors with values between 0 and 1, where the number determines how much of the information should be allowed to pass through. Cell state updates are based on addition; therefore, backpropagation does not result in a large chain rule product, thus mitigating the problem with vanishing gradients.

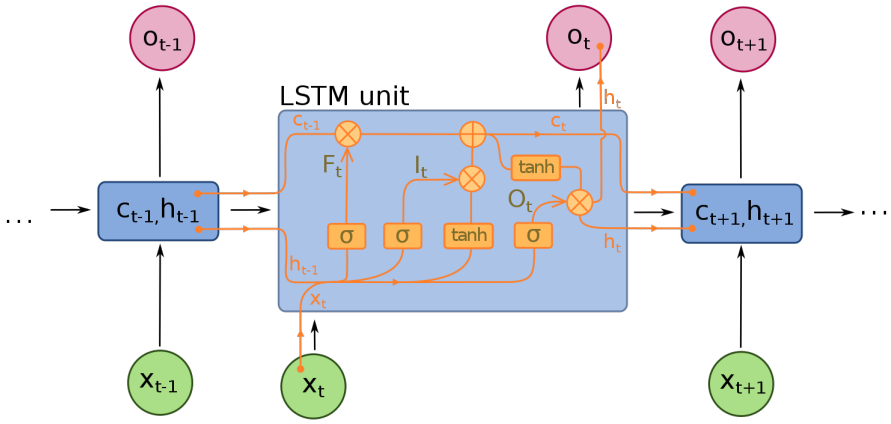


Figure 2.6: LSTM architecture. Illustration by François Deloche, licenced under CC BY-SA 4.0

2.5.2 Sequence Learning Architectures

Character-Level Language Models

Speech and language processing tasks, such as speech recognition, machine translation, and text prediction, employ language models which may be based on RNNs [53]. Text prediction (or language modeling) in particular is often done using recurrent neural networks and can be applied both at the word-level [54] and character-level [55], [56] alike. Although word-level models have been shown to produce better results in text prediction than character-level models [40], the latter is more relevant when it comes to sequence generation since it produces original output by allowing the network to invent novel words and strings. After being trained on a sufficiently large amount of text, the RNN language model uses a sequence of preceding characters to calculate the probability distribution of the following characters, thus constructing a new text one character after another. Character-level models are of special interest in this thesis because they are directly applicable to music generation tasks since symbolic music data can be modeled in the same way as text, with one character token representing a single note. Such architectures are able to generate monophonic melodies [31], [57] and chord progressions [58].

Sequence to Sequence Models

In applications such as machine translation, it is preferable to employ a model that can return a target sequence given an input sequence. This can be achieved with encoder-decoder architectures using RNNs [59], [60]. They are implemented using two separate RNNs: an encoder and a decoder (Figure 2.7). The input sequence is encoded into a fixed-length state vector which is used as the initial state in the decoder. This results in an output conditioned on the input sequence. As with the standard RNN character-level language model, this architecture can also be used for musical purposes, such as generating harmonies to an input melody [34].

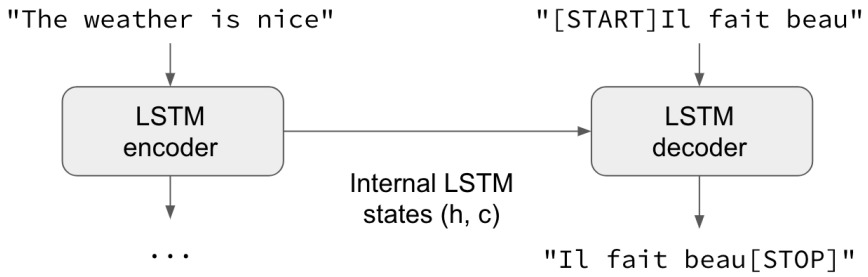


Figure 2.7: An example of an RNN encoder-decoder architecture used to translate English sentences into French. The input sentence is encoded into a fixed-length state vector that the decoder uses as its initial state when generating the output sentence. Illustration by Francois Chollet [61]

Chapter 3

Design and Implementation

This chapter describes the design and implementation of the instrument. The physical design is presented first with descriptions of the hardware and functions of the two instrument prototypes created for this thesis. It is followed by a description of the generative model used to generate melodies, the datasets and pre-processing, training of the generative model, and the concepts of sampling and playback in the instrument. Finally, an explanation of how the performer can interact with the instrument is provided. A video of the instrument with a demonstration of the controls is available on YouTube [1], and the Python scripts used in the instrument are available on GitHub [2].

3.1 Design Requirements

Based on the background outlined in Chapter 2, four design requirements were set for the instrument:

- **Interactive music generation**
  Many previous examples of music systems with neural networks focus on offline generation of music or creating musical scores. Therefore, one of the main points of focus in designing the instrument was to construct a system that is interactive. This requires that the machine learning algorithm is able to quickly respond to the user's inputs and generate music in real-time.

- **Feeling of control**
  The system is intended for users with little to no experience to provide an opportunity to play music without a considerable investment of time required to learn a traditional instrument. Because of this, the instrument must be able to generate music as well as provide a feeling of control and ownership over the musical output to the user.

[1] https://www.youtube.com/watch?v=ya4gcIvtaEE
[2] https://github.com/edrukar/intelligent_instrument

- **Self-contained**
  Most systems using machine learning to generate music require powerful computers to do so. A self-contained system on an embedded device comes with several advantages such as portability and lower maintenance requirements, as discussed in section 2.2. The performer will not have to rely on desktop computers or external hardware.

- **Inexpensive**
  The system should be easy to reproduce without investing money on expensive hardware.

3.2 Prototype 1

The first prototype (Figure 3.1) was the first attempt at creating a self-contained instrument during the earlier stages of the thesis work. It was assembled to be presented in a paper written for the NIME 2019 conference [62].

Figure 3.1: A close-up view of the first prototype. The instrument has a built-in speaker and two knobs to control volume and sampling 'temperature'.

3.2.1 Instrument Controls

The prototype has two controls: Sampling 'temperature' and volume, with sampling temperature being the most important for demonstrating the effect of adjusting parameters to shape the music and giving a feeling of control over the generated melody. Figure 3.2 shows an overview of the system.

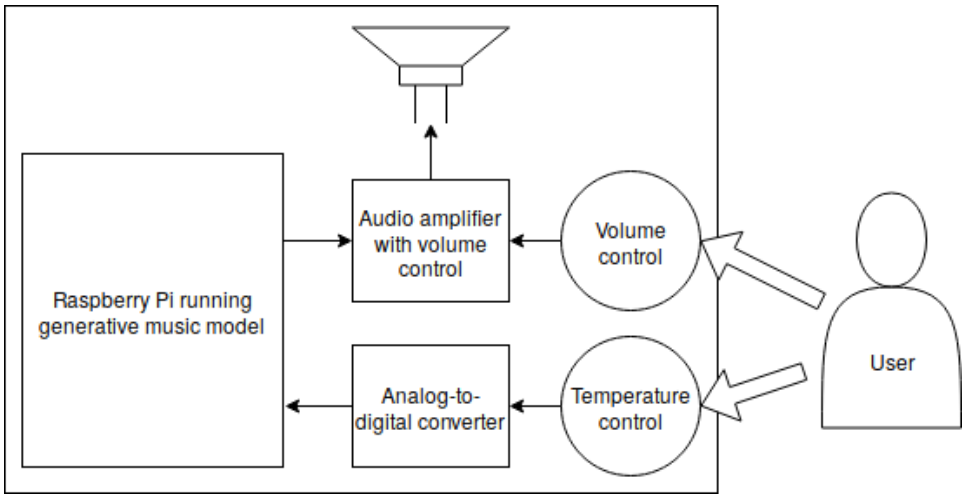


Figure 3.2: System diagram of the first prototype. A generative model runs on a Raspberry Pi single-board computer. The audio output goes through an amplifier with a potentiometer volume control and is played back on the built-in speaker. An analog-to-digital converter reads the voltage across a second potentiometer to control the sampling temperature. The user can interact with the system by adjusting the two potentiometers.

Sampling Temperature

Implementing a control for sampling temperature was chosen to be able to manipulate the output of the generative model and explore the effects this will have on the musical output. The sampling temperature is a parameter that can be adjusted to control the randomness of predictions. The model is more conservative at low temperatures, making it less likely to sample from unlikely notes, while higher temperatures allow the model to choose notes with a lower probability more often. Predictions with higher temperatures are more diverse, but can also have more mistakes. This is discussed further in section 3.8.1.

Volume

Volume control is an essential part of any music system. The volume is adjusted by using a variable voltage divider circuit to control the amplitude of the audio signal before the amplifier stage. The voltage divider is created using a potentiometer. Because the human perception of volume is logarithmic, not linear, a potentiometer following a logarithmic control law [63, p. 219] is used.

3.2.2 Hardware

Figure 3.3 shows the hardware inside of the prototype. The plastic encasement is from a local hardware store, modified to fit the hardware and painted using black spray paint. It contains the following components:

- A single-board computer to run the software
- Two potentiometers to control temperature and volume
- An analog-to-digital converter to read the potentiometers
- An audio amplifier
- A speaker

Figure 3.3: The hardware inside the enclosure. Hardware components consist of a Raspberry Pi Model B+ single-board computer, an analog-to-digital converter, two potentiometers, an audio amplifier, and a speaker.

Single-board Computer

In order to make it self-contained, the system was implemented on an embedded device. The low-cost design requirement was based on the intention to create a system that does not require expensive hardware and is therefore easily reproducible by anyone. Most single-board computers are relatively low-priced, but their differing processing capacities imply differences in price. Four SBCs were considered as options for the instrument: Nvidia Jetson, BeagleBone Black, Asus Tinker Board, and Raspberry Pi 3 B+. Nvidia Jetson is powerful and well equipped for machine learning tasks, but was discarded due to its high cost. Similarly, BeagleBone Black, equipped with the Bela expansion board, and Asus Tinker Board, with more memory, did not meet the low-cost requirement and therefore Raspberry Pi 3 B+ was chosen to run the software. Since Raspberry Pi is aimed at promoting education in computer science [23],

besides its low cost, it has a large community providing resources for both novices and expert programmers. It is also officially supported by TensorFlow framework [64], which is an open-source platform for machine learning [65].

Potentiometers

A potentiometer (Figure 3.4) is an adjustable variable resistor with three terminals that functions as a voltage divider where voltage output is determined by the position of the 'wiper' [66]. This makes it an ideal component for controlling the sampling temperature since the voltage can be mapped onto a continuous temperature value. The same is true for the volume control, where a variable voltage divider can be used to control the amplitude of the audio signal.

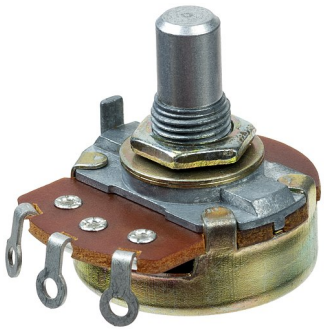


Figure 3.4: A standard potentiometer with three terminals. The voltage output is changed by adjusting the 'wiper' (in this case, a rotating shaft). Photo by Evan Amos, [Public domain], via Wikimedia Commons.

Analog-to-digital Converter

An analog-to-digital converter (ADC) converts the analog voltage signal measured across the potentiometer into a digital signal. An ADS1115 [67] ADC is used in the prototypes because it has four channels, allowing it to read multiple potentiometers separately, and an I²C interface, making it easy to control it with the Raspberry Pi. There is also a Python library available for this particular device provided by Adafruit industries.

Audio Amplifier

The audio signal is taken from the Raspberry Pi's 3.5 mm jack connection. However, this signal is not strong enough to drive the speaker. Therefore, a PAM8302 [68] 2.5 W mono audio amplifier is included to amplify the signal.

Speaker

The instrument has a built-in speaker to make it completely self-contained. The current prototype speaker was salvaged from an old monitor due to its size, which is ideal for the encasement. The sound quality was not a top priority; therefore, although it is not high, it is considered acceptable for the purposes of the current work. In addition, the built-in speaker can be easily disconnected and exchanged for an external speaker or headphones.

3.3 Prototype 2

The ability to control multiple parameters of the music increases its complexity and enriches the musical experience. Since the first prototype had to be assembled to meet the deadline of the article for the NIME conference, it has a limited number of features. Even though it can demonstrate the concept of manipulating musical predictions in real-time, more functionality was necessary to make it a complete and usable instrument. The second prototype (Figure 3.5) builds on the same principles as the previous version but includes additional hardware and software to allow for several additional instrument controls.

Figure 3.5: The second prototype. This version of the instrument has five knobs to control the sampling temperature, playback tempo, instrument sounds, volume and to switch between multiple generative models. It also has an integrated display in addition to the speaker.

3.3.1 Additional Instrument Controls

Three additional controls were added to the second prototype: generative model selection, tempo control, and instrument sound selection. An overview of the complete system is illustrated in Figure 3.6.

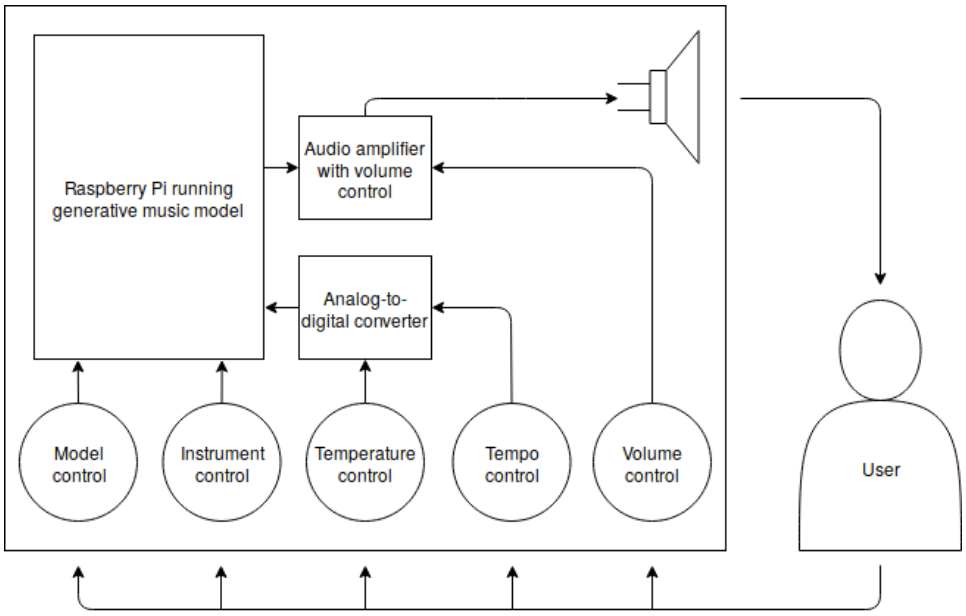


Figure 3.6: System diagram of the second prototype. A generative model runs on a Raspberry Pi single-board computer. The audio output goes through an amplifier with a potentiometer volume control and is played back on the built-in speaker. An analog-to-digital converter reads the voltage across two other potentiometers to control sampling temperature and playback tempo. Two rotary encoders are used to switch between multiple generative models and instrument sounds. A display shows which generative model is currently active.

Generative Model Selection

A way to make the device more versatile is implementing the ability to generate melodic lines in multiple musical styles and quickly switch between them during a performance. Users can choose and activate different trained models at any time during playback by using one of the rotary encoders. The weights of each model are assigned an index number, which can be incremented or decremented by turning one of the rotary encoders. The new model is activated by pressing the push-button on the rotary encoder, which loads the chosen weights into the LSTM network and resets the cell states of the network before resuming sampling and playback. There is no specified limit to how many different sets of weights the instrument can support, so users are free to train models on datasets in their preferred genres and include them in the instrument. The current system has three generative models trained on different datasets.

Tempo

The tempo is a fundamental part of any musical piece on a par with melody, harmony, and rhythm, among others. The same melody played at different tempos can make it sound anything from somber to comical. Therefore, a third potentiometer was added to the prototype to control the playback tempo. The voltage signal sampled by the ADC is converted into a value of milliseconds used for time delays corresponding to beats per minute (BPM) value.

Instrument Sound Selection

Having the ability to switch between different sounds makes the instrument much more flexible by giving it the potential to adapt to the musician's current needs, for example, to play in an ensemble with other instruments during a performance or in a studio setting. The instrument sounds are from the `FluidR3` SoundFont, which contains 128 instrument sounds, following the General MIDI sound set [69]. Some of the instrument producing low volume outputs were discarded, leaving a total of 56 instruments to choose from. Users can switch between instrument sounds using one of the rotary encoders.

3.3.2 Additional Hardware

Figure 3.7 shows the inside of the second prototype. The plastic encasement was modified and painted in the same way as for the first prototype. The additional hardware used in order to support the added functionality of the instrument includes:

- A third potentiometer to control playback tempo
- An OLED display to show which generative model is active
- Two rotary encoders to switch between generative models and instrument sounds

Rotary Incremental Encoders

When switching between generative models and instrument sounds, it is convenient to have a control that measures discrete steps instead of continuous values. Rotary incremental encoders [70] are well suited for this type of task. They are electromechanical devices that resemble the potentiometer in Figure 3.4, but instead of outputting a variable voltage value controlled by the rotating shaft position, they generate pulses in response to incremental movements of the shaft. Each time the shaft is turned one step, a counter value is updated, serving as an index for models or instrument sounds.

Figure 3.7: The hardware inside the enclosure of the second prototype. The system is implemented on a Raspberry Pi Model B+ single-board computer. Other components include an analog-to-digital converter, rotary encoders, potentiometers, an audio amplifier, speaker, and an OLED display.

Display

Since the second prototype supports multiple generative models, it is convenient to have a display showing which model is active. A 0.96" 128x64 OLED display with an SSD1306 driver [71] was implemented for this purpose. This display was chosen because it is easily controlled via I²C, and Adafruit provides a Python library for the SSD1306 driver. The display (Figure 3.8) shows which model is active and which is the next to be activated.

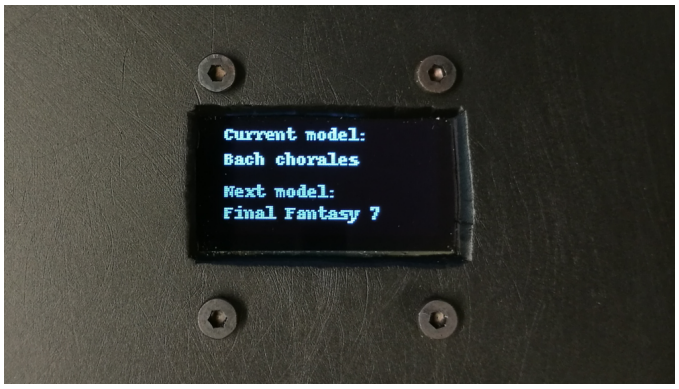


Figure 3.8: The display on the instrument. Lines 1 and 2 show the active generative model. Lines 3 and 4 show the next model to be activated.

3.4 Generative Model

The core of the instrument is a generative model that is able to generate a continuous sequence of musical notes. There were two main requirements that affect the choice of architecture: it must be simple enough to make note sampling in real-time possible (sampling time must not exceed the note length), and it has to remember what it has played previously to achieve coherency in the generated sequences. Language models using recurrent neural networks, and character-level models in particular (discussed in section 2.5.2) are well fit for this task. They were chosen due to their relative simplicity and ability to predict single characters (or notes) at each sampling step.

3.4.1 Architecture

The generative model (Figure 3.9) uses a character-level language model architecture with recurrent neural networks, as described in section 2.5.2. It is implemented in Keras [72], a high-level Python API for neural networks, running on top of Tensorflow. The model consists of two LSTM layers with 256 units each, an embedding layer on the input, and a dense layer with a softmax activation function on the output. The embedding layer transforms the input integer representation of notes into vectors of fixed size that the two LSTM layers can process hierarchically. The dense layer with softmax activation then projects the output from the LSTM layers back into probability distributions over possible note values.

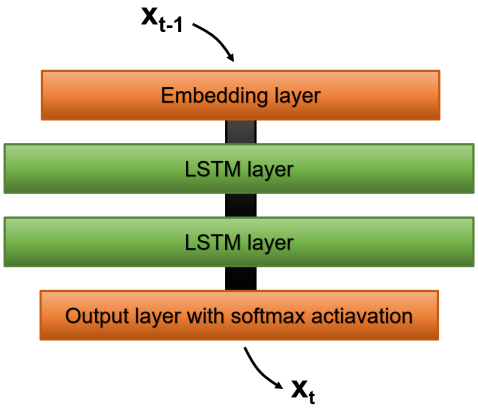


Figure 3.9: The architecture of the generative model. It contains an embedding layer, two LSTM layers with 256 units each, and a dense layer on the output. By returning the output of the RNN to the input, a continuous stream of notes can be generated.

A common network size for monophonic music generation is three LSTM layers with 512 units each [5], [31]. However, since the model runs on an embedded device with limited processing power, it is also necessary to account for sampling times, which will increase with the number of layers and units. At maximum playback tempo (120 BPM), one sixteenth note lasts 125 ms. To maintain a constant tempo, the time it takes to sample a single note from the model must always be lower than 125 ms. Table 3.1 shows the average sampling times with different network sizes on the Raspberry Pi. The results are based on an experiment where 500 samples were drawn from each of the network architectures.

Table 3.1: The average time it takes to sample one note from networks of different sizes on the Raspberry Pi, calculated from 500 samples. Size 3x512 was not measured since 2x512 had already failed the timing requirements.

Network size	**Sampling time (ms)**
2x256	60
3x256	82
1x512	63
2x512	181

As can be seen from the table, multiple layers of 512 units give sampling times that are too high. Although one layer of 512 units is an option, deeper networks with fewer units in each layer have been shown to perform better than fewer layers with more units [73]. Two layers of 256 units are used, even though three layers also meet the time requirements. Some experimentation with both architectures indicated that the models with

two layers were better at capturing the musical structure of the datasets used in this thesis.

3.5 Datasets

There are three datasets from different music genres: chorales, traditional Irish music, and video game music. The chorales and traditional Irish music are available as part of a corpus of freely distributable music contained in the Music21 toolkit [74], while the video game music is a collection of MIDI files downloaded from the *Midi Shrine* [75]. These datasets are used in part because of accessibility, but also to test datasets of varying quality. Audio samples from the datasets presented here are available on Zenodo [3].

3.5.1 Bach Chorales

The first dataset is a set of 405 chorales composed by Johann Sebastian Bach. They consist of four separate voices: soprano, alto, tenor, and bass (SATB). Although it is the melody sung by soprano that is often perceived as the most prominent, and the three lower voices have a more supporting role by creating harmonization, all the voices can act as monophonic melodies on their own. Most of the time, they follow the same rhythmic patters, so combinations of different voices should not be a problem for the coherency of music generated by a model trained on different voices. It is therefore expected that the Bach dataset is well suited for training the generative model. Each voice is used as a single melody during training, giving a total of 1620 training examples of monophonic melodies. An example from the dataset is shown in Figure 3.10.

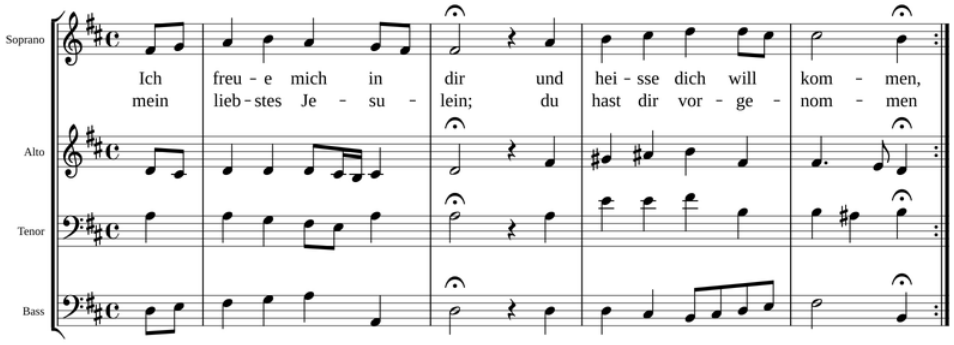


Figure 3.10: Part of a Bach Chorale with four-part harmony for SATB vocalists. Chorales are typically divided into phrases, where the end can be identified by the pause signs in all four voices, as seen in measure three.

[3] https://doi.org/10.5281/zenodo.3333505

3.5.2 Ryan's Mammoth Collection

The second dataset is also taken from the Music21 corpus. It is a set of traditional Irish music pieces titled *Ryan's Mammoth Collection* [76], containing 1059 pieces of relatively short length. All the melodies are monophonic, which is an advantage since they do not rely on harmonization from other instruments. For this reason, Ryan's Mammoth collection is expected to be the dataset best suited for the generative model. Most of the tunes have a majority of sixteenth notes (Figure 3.11), making the melodies from this dataset sound much faster compared to the Bach chorales, which consist mostly of quarter notes and eighth notes.

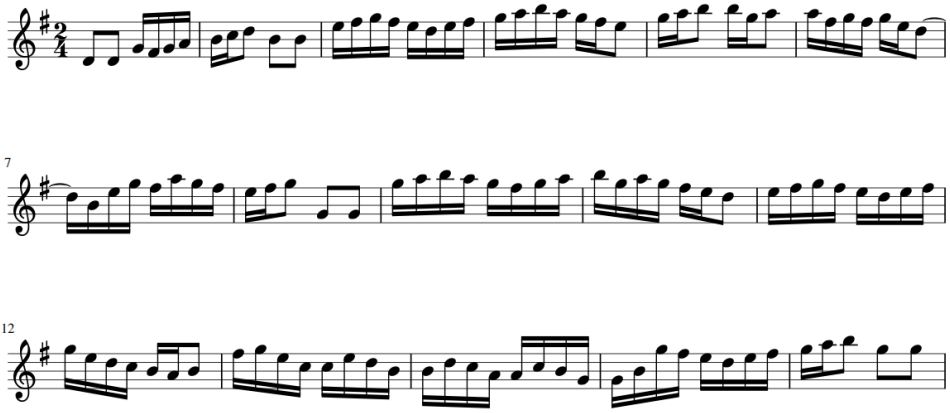


Figure 3.11: An example of a tune from Ryan's Mammoth Collection. The songs in this dataset are short, monophonic, and generally quite fast, with a majority of sixteenth notes.

3.5.3 Final Fantasy 7

The final dataset is comprised of music from the popular role-playing video game, Final Fantasy 7. It is included as an example of how users can download and train their own datasets from the internet. Even though it contains only 85 music pieces, it is still much larger than the two previous datasets, since the music pieces are much longer, and they all contain multiple instrument parts. There is a total of 1008 melody lines after separating the instruments. This dataset is less ideal for the purpose of training a network on monophonic melodies because the different instruments rely on each other much more than in the previous two datasets. An example of a score is illustrated in Figure 3.12. Note that the topmost violin plays the melody, while the bottom three violins play repetitive patterns to support the melody, and many of the other instruments act only as chord harmonizations when played together.

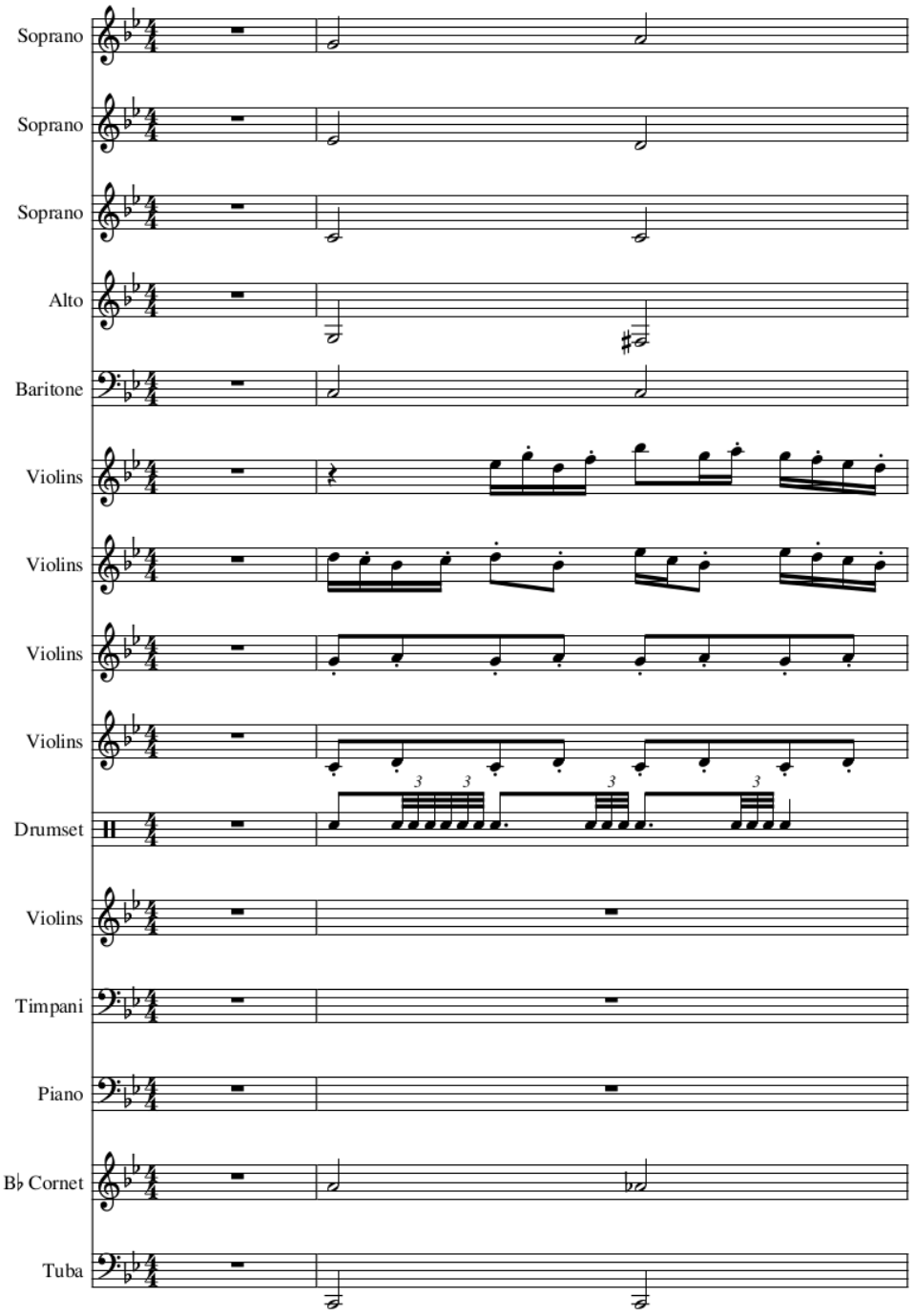


Figure 3.12: Part of a score from the Final Fantasy 7 collection. There are multiple instrument parts, some of which are silent for longer periods of time. Note also that the topmost violin plays the melody, while the bottom three violins play repetitive patterns to support the melody, and many of the other instruments act only as chord harmonizations when played together.

3.6 Data Pre-processing

Some pre-processing is necessary to transform raw MIDI data into a representation that can be used to train the recurrent neural network.

3.6.1 Note Encoding

The system encodes music using 1D vectors of integers, which is a common type of note encoding for music generation with RNNs [57], [77]. The integers are in the range of 0–129, as shown in Figure 3.13. The encoded interpretations are as follows:

- 0–127: NOTE_ON
- 128: NOTE_OFF
- 129: NO_EVENT

0–127 are pitches from the standard MIDI format, 128 tells the system to stop the note that was playing, and 129 represents no change. Each integer event has a duration of one sixteenth note, which is one sixteenth of a whole note (the length of one bar in a piece of music with a $\frac{4}{4}$ time signature). When encoding music in Python, MIDI files are converted into stream objects with the Music21 toolkit, and then to integer vectors.

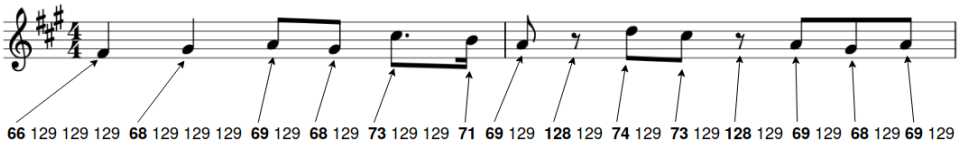
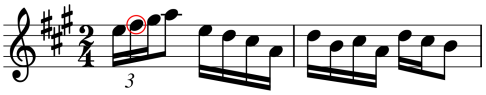
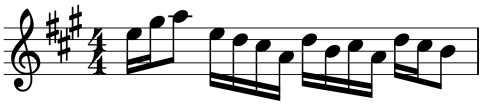


Figure 3.13: Integer representation of notes, each with a duration of one sixteenth note. The numbers 66, 68, 69, 71, 73 and 74 represent MIDI note pitches. 129 means no change, so the previously played note will be held until either a new note is played, or a value of 128 turns the note off.

Some simplifications are made when encoding MIDI into the integer representation: chords are simplified to only the highest notes, and complex rhythms are simplified to sixteenth note versions. This means that some notes are lost in the process. In Figure 3.14 we can see that one note from a sixteenth note triplet is lost. A sixteenth note triplet is three sixteenth notes played with the same duration as two sixteenth notes (or one eighth note). Due to the sixteenth note quantization, 32nd-notes are also lost, as seen in Figure 3.15.

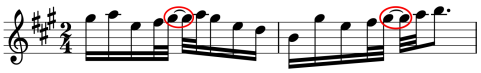


(a) Original melody with triplets.

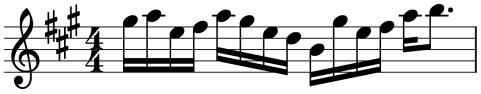


(b) Melody after encoding.

Figure 3.14: One note from the sixteenth note triplet (marked with red (a)) is lost during the encoding (b).



(a) Original melody with 32nd-notes.



(b) Melody after encoding.

Figure 3.15: 32nd notes from the original melody (a) are lost during the encoding (b).

3.6.2 Transposition

The act of transposition [78], visualized in figure 3.16, involves shifting a musical sequence to a higher or lower pitch. To the human ear, transposing a sequence will not make much of a difference; as long as the intervals, or relationships between notes in a sequence, are the same, we will recognize the sequence as the same as well. For example, the sequences $(C \quad E \quad G \quad C)$ and $(A \quad C\# \quad E \quad A)$ will, by the human listener, be categorized as the same sequence. The same is not true for the RNN. Unlike convolutional neural network (CNN) architectures, which can have invariant properties along multiple directions [79], an RNN is not spatially invariant (or in this case, note invariant) [80]. The word "music" is very different from "nvwjd", where each character is shifted up one step. Similarly, the sequences $(4 \quad 8 \quad 11 \quad 16)$ and $(1 \quad 5 \quad 8 \quad 13)$, which are the integer representations of the previous sequences, will produce completely different outputs. The goal for the RNN is to learn the relative relationships between the notes in the melodies to retain structural similarities to the original pieces and achieve clearer stylistic patterns. Due to a limited number of training examples, it was decided to transpose all of them to the same musical key. The Bach- and Ryan's Mammoth Collection datasets have melodies in 19 and 15 different keys, respectively. All major keys were therefore transposed to C major, and all minor keys were transposed to A minor prior to training.

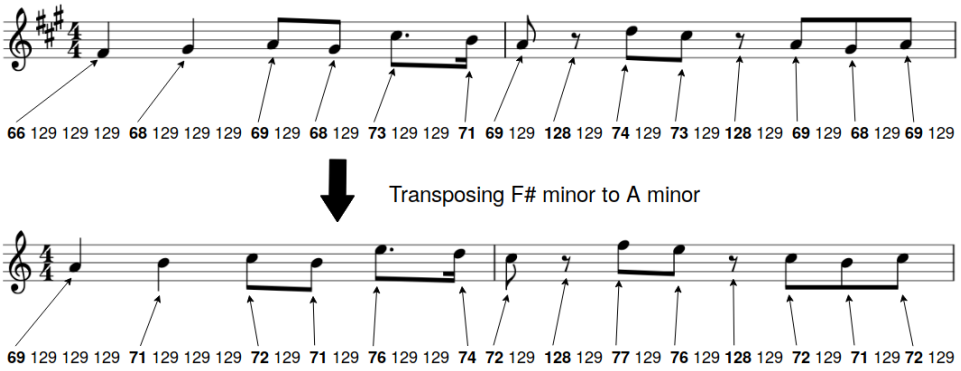


Figure 3.16: Transposing a piece of music up three half steps from the key of F# minor to the key of A minor. The intervals between the notes remain the same, so a human listener will hear the same melody, but in a different pitch.

3.6.3 Creating Subsequences

Songs from the datasets were split into shorter subsequences of 33 notes, where the first 32 notes served as the input sequence, and the last note served as the target value. There is some overlap in the subsequences, as illustrated in Figure 3.17. For each subsequence, the window slides nine steps. Because the songs have varying lengths, the shorter stride makes sure that less data is lost. The number of subsequences for each dataset is listed in Table 3.2. During training, 10 percent of the sequences were used as validation sets.

```
[ 69 129 71 129 72 129 129 129 71 129 69 129 68 129 129 129 ]
[ 69 129 71 129 72 129 ]
              [ 72 129 129 129 71 129 ]
                           [ 71 129 69 129 68 129 ]
```

Figure 3.17: Splitting a note sequence into three shorter subsequences. In this example, the subsequences have a length of six notes, and the window moves with a stride of four steps.

Table 3.2: The number of training subsequences and validation subsequences for the three datasets.

Dataset	Total	Training	Validation
Bach chorales	35292	31762	3530
Ryan's Mammoth Collection	19229	17306	1923
Final Fantasy 7	51075	45968	5107

3.6.4 Removing Empty Sequences

One challenge that can often be encountered with datasets of more complex music downloaded from the internet, such as the Final Fantasy 7 dataset, is that some instrumental lines can often have long periods of silence, which means that NO_EVENTs are over-represented in certain melody lines. To mitigate this problem, subsequences that contain only NO_EVENTs were removed from the Final Fantasy 7 dataset. This reduced the number of subsequences from 77023 to 51075. Another challenging aspect of such music is percussive lines, which tend to repeat single notes over and over. No attempts were made to remove these instrument tracks at this stage, but it could be addressed in future work.

3.7 Training

Training of the LSTM models was done using gradient descent with a batch size of 64 and sparse categorical cross-entropy as the loss function. Adam [81] was used as the optimizer, with the parameters provided as default values in Keras. These are the same as in the paper that originally proposed the Adam optimizer. Weights were saved locally using the Keras callback function `ModelCheckpoint` so that they can be loaded into the generative model.

As can be seen in Figure 3.18 and 3.19, the training loss starts to converge after approximately 40 epochs in the Bach and Ryan's Mammoth Collection models, while the Final Fantasy 7 model (Figure 3.20) begins to converge after approximately 20 epochs. However, only the Bach- and Ryan's Mammoth Collection models converge to approximately zero, while the Final Fantasy 7 model stabilizes around 0.1. It is possible that the number of parameters in the network is high enough for the two smallest datasets but too low for the Final Fantasy 7 dataset. The models also begin to overfit after 20 epochs for the Bach and Ryan's Mammoth Collection datasets, and ten epochs for the Final Fantasy 7 dataset, as indicated by the increasing validation loss. This is not necessarily a bad thing. Predicted music should sound pleasing to the human ear, and since overfitting means that

the model has begun, to some extent, to memorize the input sequences, it can be expected that the musical output will be better at following the structure of the training examples. Although overfitting is usually undesirable, the effect can be mitigated by the adjustable sampling temperature. As the user increases the temperature, predictions will begin to deviate from the possible learned training examples.

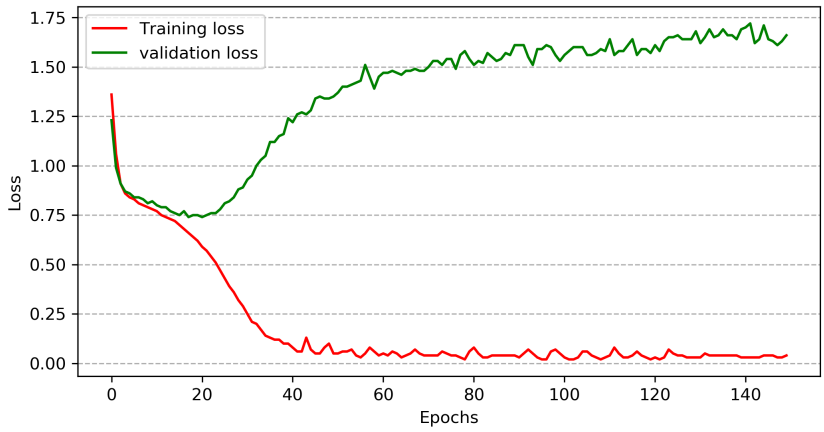


Figure 3.18: Training loss when training the LSTM network on the Bach chorales dataset. The model begins to converge after approximately 40 epochs. The increasing validation loss after 20 epochs indicates that the model is beginning to memorize the input sequences.

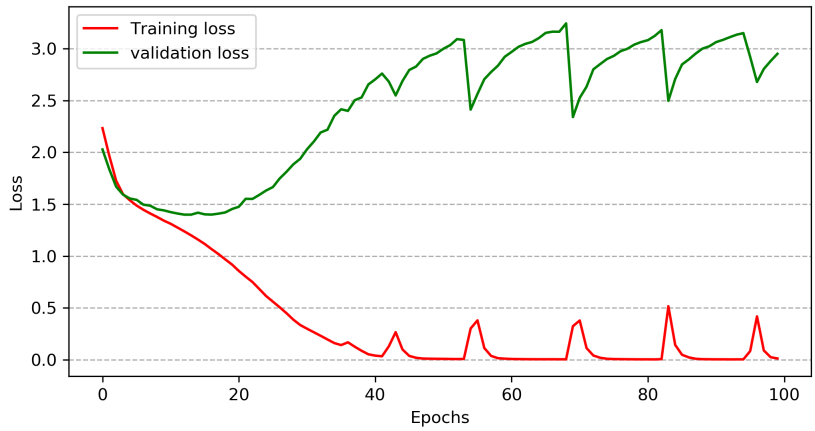


Figure 3.19: Training loss when training the LSTM network on the Ryan's Mammoth Collection dataset. As with the training of the Bach chorales model, this model also begins to converge after approximately 40 epochs. Similarly, this model begins to overfit quite early, but the validation loss is somewhat higher than for the Bach model.

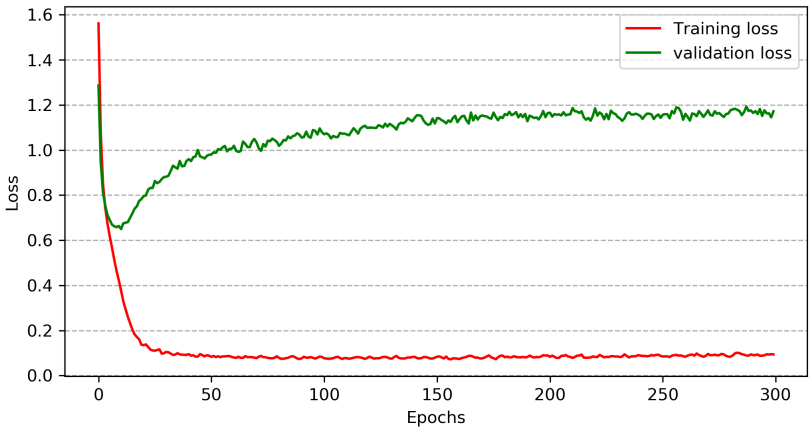


Figure 3.20: Training loss when training the LSTM network on the Final Fantasy 7 dataset. Unlike the Bach- and Ryan's Mammoth Collection models, which converge to approximately zero, the Final Fantasy 7 model stabilizes at a training loss of around 0.1. However, the validation loss is a little lower than for Bach and Ryan's Mammoth Collection.

Adding Dropout

Since overfitting can be mitigated by adjusting the sampling temperature on the instrument, it was not considered a problem. Therefore, optimization of the training was not central to the current work. Nonetheless, an experiment using dropout between the recurrent layers was performed. The concept of dropout is to randomly remove units and their incoming and outgoing connections from the network during training, and it has been shown to be an effective technique to reduce overfitting [82]. A relatively low dropout of 0.2 was applied. It reduced overfitting by a small amount, but it also affected the melodic structure of predictions. Musical quality in the current work is deemed to be more important than optimization of the training; therefore, training with dropout was excluded.

3.8 Sampling and Playback

Sampling from the model returns a vector of probability distributions over all possible note values. From this distribution, the most probable next note can be found based on previously predicted notes in the sequence, as illustrated in Figure 3.21. Two processes—one for sampling, and one for playback—run in parallel. When sampling, the model is given a seed note as a starting point that it uses to predict the next notes in the sequence. At the start, the default value is set to 60, which is a middle C. After each prediction, values of 128 (NOTE_OFF) and 129 (NO_EVENT) are filtered out, and the last element in the sequence is stored and used as the seed for the next prediction. This allows the instrument to play a continuous melody. The playback process sends notes over an internal

MIDI connection for synthesis via the `Timidity++` software synthesizer and `FluidR3` SoundFont. Each time it begins to play a sample, the sampling process is notified and starts to draw a new sample from the model. This allows the system to predict musical sequences in real-time. Temperature and tempo values are read from the ADC before each sample so that the controls take immediate effect.

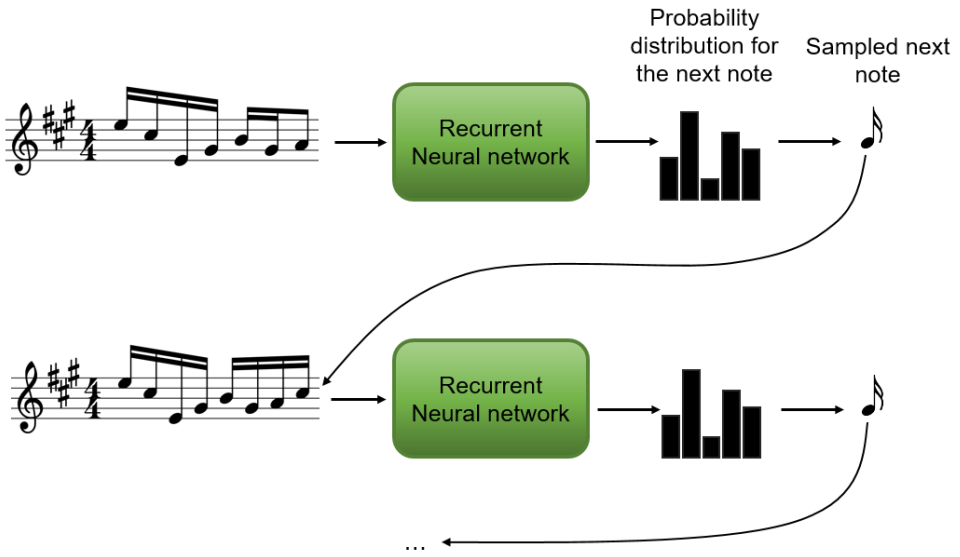


Figure 3.21: Prediction of note sequences. Sampled notes are added to the output sequence, which is fed back to the input to be used for further predictions.

3.8.1 Temperature

There are several ways to choose the next token in a sequence. The simplest of them is *greedy sampling*, which assigns a certain token within a probability distribution the probability of 1 and the remaining tokens a probability of 0. This approach will always choose the most likely next token and output sequences that are repetitive and predictable. Stochastic sampling, or sampling process that includes more random probabilities, can produce more interesting results [83]. A parameter called *temperature* is used to control the amount of randomness, or entropy, in the probability distribution when sampling stochastically. A temperature factor of 1 has the same effect as applying regular softmax (eq. 3.1) to the output of the dense layer of the network. When the temperature T is changed, the distribution is reweighted by scaling each element in the output vector z, resulting in a new distribution q with a different entropy (eq. 3.2).

$$softmax(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{3.1}$$

$$q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{3.2}$$

By increasing the temperature, the entropy of the distribution will rise, allowing more variation in the predicted sequences. Sufficiently high temperatures, however, will in practice ignore the predictions of the trained model and choose completely random outputs instead, as each element in the distribution will have approximately the same probability of being chosen. Similarly, a temperature below one will make the predictions more greedy. The current system uses a temperature range of 0–10. Having a maximum temperature higher than ten does not seem to be useful because the resulting samples are almost completely random at this temperature value.

3.8.2 Tempo Control

When a note is played, it is held for a time equal to the note length, corresponding to the chosen BPM. The note length is specified in milliseconds: 60,000 ms (one minute) / Tempo (BPM) = note length in ms for quarter-notes. Dividing this result by four yields the length of sixteenth notes that are used in this system. Users can adjust the lengths of the notes using one of the potentiometers on the instrument. The minimum and maximum note lengths are 125 ms and 1125 ms, corresponding to a BPM range of 13–120.

3.9 User Interaction

When the instrument is turned on, it automatically starts to play a continuous monophonic melody sampled from the generative model. The user can see which model is active on the display, and scroll through a list of the models with the knob attached to the rotary encoder on the bottom left. The next model on the list is shown also shown at the bottom of the display, and is activated by pressing the knob down, which triggers the push-button integrated in the rotary encoder. If the current instrument sound is not preferred, it can be easily changed by turning the other rotary encoder. The instrument sound is updated immediately with each rotation increment. To have more diversity in the generated melodies, the user can adjust the sampling temperature by turning the temperature knob. The diversity ranges from predicting only the most likely next notes, to almost completely random note choices. In addition, the playback tempo can be increased or decreased with the tempo knob. This combination of output control parameters makes it possible to create unique performances with little effort.

Figure 3.22: A user interacting with the five control knobs on the instrument during a performance.

Chapter 4

Testing and Evaluation

The term *evaluation* can have different meanings. In NIME literature, the term is commonly used to outline the procedure for collecting information from users on how to improve a prototype, whether to determine if the device is suitable for particular tasks, or to compare several devices using some common characteristics [84]. Different interests with regards to DMI design often result in different requirements to instrument prototypes [85]. A performer may focus on playability, as well as its reliability during live performances, while the audience may only be interested in how it sounds. The designer's primary objective can instead be whether the prototype serves its intended purpose.

The evaluation in this thesis was done mainly from the designer's and the performer's perspective. The former was performed by the author with a subjective evaluation of the generative models and an assessment of whether and how the instrument meets the pre-determined design requirements. The latter involves a user study where the participants were introduced to the instrument and provided feedback by responding to a set of questions.

4.1 Generative Models

There are essentially no "wrong" choices of notes in a melody since the experience of music is highly subjective, and it usually comes down to personal preference. Because metrics for evaluating music sequences created by generative models are very limited, subjective evaluation is left as the most viable choice [86]. This section presents sample sequences generated by the three generative models, which are compared to the dataset samples in section 3.5 by visual examination of the sheet music and by listening to the audio. All audio samples are available on Zenodo [1].

[1] https://doi.org/10.5281/zenodo.3333505

4.1.1 Bach Chorales Model

Figure 4.1 shows eight measures sampled from the model trained on the Bach chorales dataset. A visual comparison of the original sheet music, a piece of which can be seen in Figure 3.10, and sheet music of the generative model, shows that the musical structure is very similar to the dataset, and it is easy to hear Bach chorale elements in the predicted melody. Particularly noteworthy is the C going up three half-steps to an F (marked in red) before creating natural pauses in the melody. The same can be observed in Bach chorales, where this stylistic feature is used to divide the melody into phrases. The interval of three half steps is of the same type as in the bass line before the phrase ends in Figure 3.10. In the author's opinion, the generated music is enjoyable with melodies of clearly recognizable musical structure.

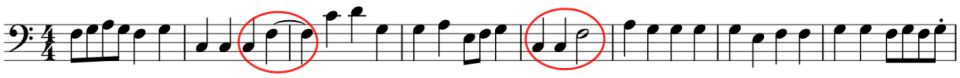


Figure 4.1: Music sampled from the Bach chorales model with the temperature set to 1.0. The predicted melody clearly contains stylistic elements from the original dataset, such as resemblance of ending phrases (marked in red).

4.1.2 Ryan's Mammoth Collection Model

The Ryan's Mammoth Collection model is also able to reproduce many stylistic elements from the original dataset, containing fast melodies with mostly sixteenth notes (Figure 4.2). Particularly notable is how measure one resembles measures nine and ten in Figure 3.11, with 14 sixteenth notes followed by one eighth note. The predicted melody sounds very much like the tunes from the dataset and is of a standard similar to the Bach chorales model in the author's opinion.

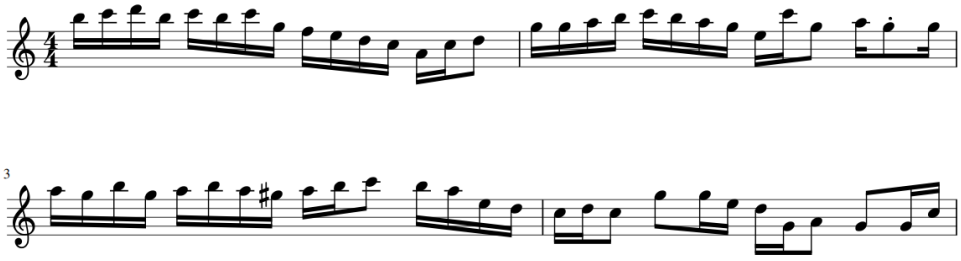
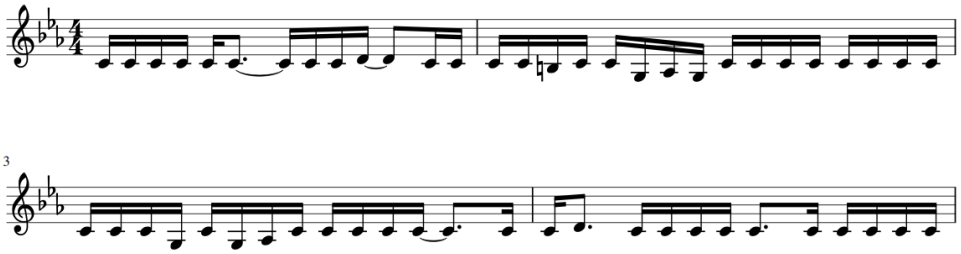


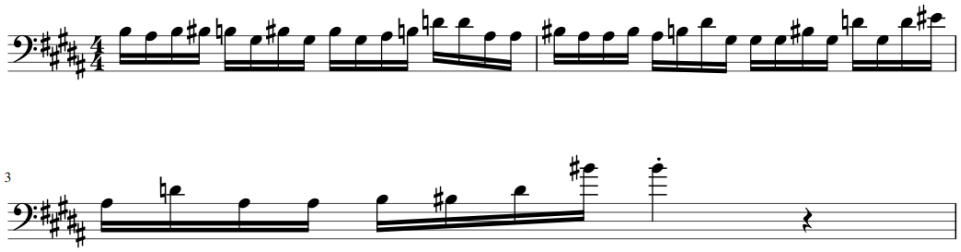
Figure 4.2: Sample from the Ryan's Mammoth Collection model, taken with a temperature set to 1.0. Visual comparison of the model output with the example from the dataset (Figure 3.11) shows that there is a clear resemblance of the musical structure. It is noteworthy how measure one resembles measures nine and ten in Figure 3.11, with 14 sixteenth notes followed by one eighth note.

4.1.3 Final Fantasy 7 Model

Two samples taken from the Final Fantasy 7 model can be seen in Figure 4.3. We can see that at a temperature of 1.0, there is little variation in the melody since the output contains mostly repeating sixteenth notes with only some minor variations. An increase in sampling temperature gives a slightly more varied melody, but the same notes tend to be repeated here as well. This can be expected from a dataset containing drums and percussion tracks because they have notation for rhythmic patterns, which do not include any change in pitch. It is more challenging to compare the musical output from this model to the original dataset, as there are many instrument voices, and the network is trained on all of them. However, the predictions in Figure 4.3b somewhat resemble the two topmost violins in the sample from the dataset in Figure 3.12. It is worth noting that all samples taken from the generative models are played at a tempo of 120 BPM in the audio files on Zenodo, whereas the original dataset example, in this case, is much slower at 57 BPM. The music generated by the Final Fantasy 7 model is, in the author's opinion, less enjoyable than that of Ryan's Mammoth Collection and Bach due to less coherent melodies and more repetitiveness.



(a) Temperature 1.0



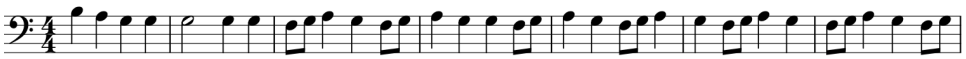
(b) Temperature 2.0

Figure 4.3: Two samples drawn from the The Final Fantasy 7 model. This model produces highly repetitive and uninteresting results when the temperature is set to 1.0 (a). Increasing the temperature setting to 2.0 (b) results in somewhat more interesting musical output, but the repetitiveness is still an issue.

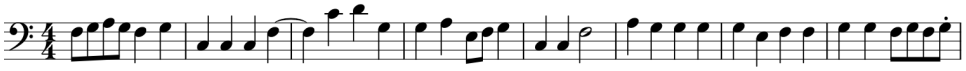
4.1.4 Effects of Different Sampling Temperatures

Figure 4.4 shows a set of samples drawn from the generative model trained on Bach chorales model at different temperatures. With the temperature

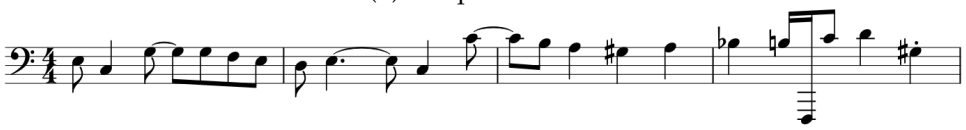
set to 0.1, which is quite close to a greedy sampling strategy, the melody has a tendency to become very repetitive. The melody begins to sound quite close to the training examples when setting the temperature to 1.0. Increasing the temperature to 3.0 introduces more randomness to the melody while still retaining the general structural elements from the training examples. This gives the resulting melody a more "jazzy" sound. When the temperature is set to 5.0, any resemblance to the training examples is difficult to hear. Higher temperatures than 5.0 produce melodies that are virtually random. This shows that the optimal temperature values for retaining the musical structure of the training examples while still allowing for some creativity are between 1.0 and 3.0.



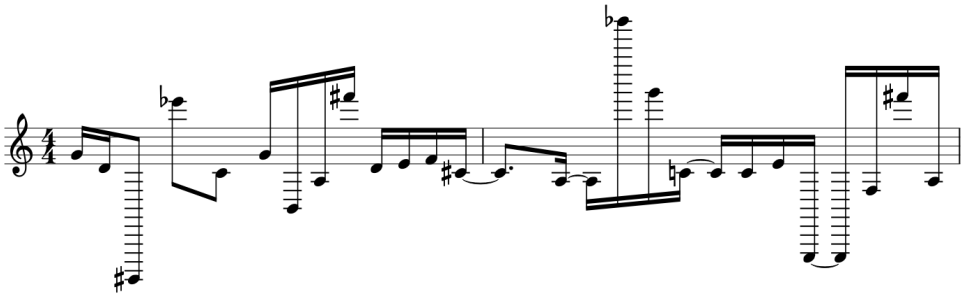
(a) Temperature 0.1



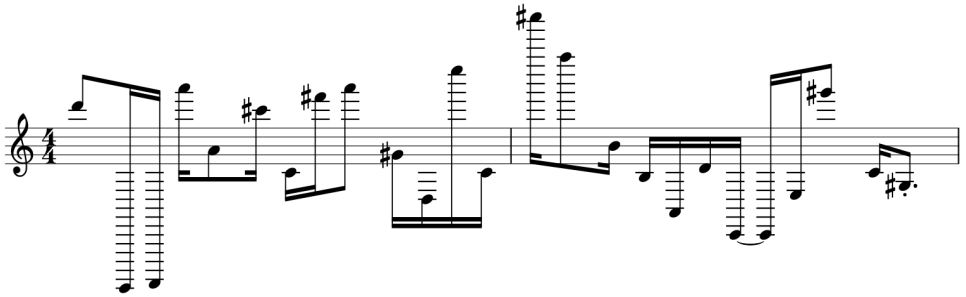
(b) Temperature 1.0



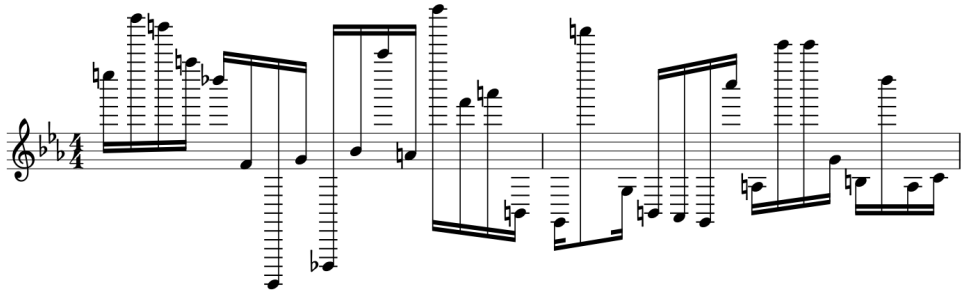
(c) Temperature 3.0



(d) Temperature 5.0



(e) Temperature 8.0



(f) Temperature 10.0

Figure 4.4: Sampling the generative model trained on Bach chorales at different temperatures. At temperature 0.1 (a), the melodies are highly repetitive. Temperatures of 1.0 to 3.0 (b and c) generate melodies that sound close to the training examples. At 5.0 and above (d, e and f), the results begin to sound virtually random.

4.2 System Design

This evaluation of the system design is based on the author's experience with the instrument, with regards to the four design requirements outlined in section 3.1. This assessment is considered to be from both the designer's and performer's perspective since the author fills both roles. However, the focus lies on the actual design, and whether and how the instrument is able to perform the tasks it is intended for.

4.2.1 Evaluation based on Design Requirements

Interactive Music Generation

The instrument is capable of predicting and playing samples from the generative model in real-time. The value of the temperature and tempo is read from the ADC before each prediction, so it is possible to change these parameters and hear the effect almost instantly. The only exception is at the slowest tempo, where a sixteenth note lasts 1.1 seconds. In this case, the performer has to wait for the sample to finish playing before the changes take effect. However, this is not seen as a big concern, and it can likely be fixed by modifying the code if necessary.

Feeling of Control

All of the instrument controls affect the musical output to some degree. Switching between multiple generative models, for example, lets the user decide which musical style to use at any point during a performance. Being able to change the tempo and synth have a large effect on the sound and general feel of the musical output as well. In the author's opinion, however, the temperature is the most important control since it affects musical predictions and not only sound and playback. The instrument can follow the structure of the datasets at lower temperatures, or generate melodies that are akin to experimental jazz at higher temperatures. The author argues that having the option to control the diversity of the generated melody provides the user with a sense of ownership over the music, resulting in a more enjoyable and rewarding experience. Another important effect of the temperature is the ability to "reset" the LSTM model if it becomes overly repetitive. Increasing the sampling temperature to a higher value for a few seconds will ensure that the model includes some unlikely notes, thus creating a more random sequence at the model's input which can then be used for further predictions.

Self-contained

Implementing the system on a single-board computer and integrating all the necessary hardware for interaction and sound inside the enclosure makes the system completely self-contained. The only required external input is power through the Raspberry Pi's micro USB port. It can even be connected to a USB power bank to make it portable. This proved to be

particularly useful at the NIME 2019 conference, where no power outlets near the poster stand could be found for the demonstration.

Cost

Reducing the cost as much as possible was an important requirement in order to make the instrument easily reproducible without any major investments. The total cost of the instrument, including all of its components, is approximately 100 USD. This can be reduced further with the access to a 3D printer to create the encasement, or by buying cheaper components on websites such as eBay.

4.2.2 Experiences from NIME 2019

The instrument was presented during a poster session at the NIME conference 2019 in Porto Alegre since the article dealing with the first prototype was accepted for publication in the conference proceedings. The decision to present the second prototype was made with the desire to gain further insight into how users may interact with the current instrument. During the poster session, conference participants were welcome to test the instrument. Unfortunately, the session was too busy to take detailed notes; therefore, only general impressions were noted down after the session. The reactions to the instrument were exclusively positive and received multiple comments expressing appreciation of how the instrument is built and what it does. Many questions addressed the temperature control, and several participants thought it had something to do with the actual temperature of the room. After receiving an explanation, the majority of those who tried the instrument seemed to enjoy the temperature control the most, especially at higher values. This was expressed mostly with body language, such as nodding as in approval of the musical output, and in some cases verbal expressions of favoring the temperature parameter, such as "I love the temperature."

4.3 User Study

A user study was conducted to evaluate the instrument from a performer's perspective, which is arguably the most important [85]. The goal of the study was to see the impact the different high-level parameter controls can have on a participant's perceived feeling of control over the musical output from the instrument, and to evaluate the three trained generative models in terms of musical quality.

4.3.1 Session Overview

The sessions were split into four parts, listed in Table 4.1. Part A was an introduction with a brief explanation of sequence generation and how to operate the instrument. In the remaining parts, the participants explored the instrument and answered questions about their experience with the

instrument controls and generative models, as well as shared general thoughts about the instrument.

Table 4.1: User study: Session structure

Part	Activities
Part A	Introduction
Part B	Free play
Part C	Model evaluation
Part D	Interview

Part A: Introduction

The first part of the session was a brief introduction to the instrument. This included a quick introduction to generative models and how they can be used to create music, the three datasets the models were trained on, how the instrument works by sampling the model and playing back the notes in real-time, and an explanation of the different instrument controls. Participants were from different educational and occupational backgrounds with varying degrees of knowledge about machine learning and music, so the time spent on the introduction was assumed to vary between sessions. Nevertheless, the participants were expected to understand the basic concepts of the system within the first few minutes.

Part B: Free Play

The second part of the study addressed how the participants engage with the instrument control knobs in general, and whether they felt some degree of control over the generated music in particular.

After the introduction, participants spent some time exploring the instrument to familiarize with the instrument and its functionality to get a feeling of how it can be used to make music. It was expected that the average participant would spend approximately 5–10 minutes for this purpose. The participants were asked to tell the interviewer when they felt they had become sufficiently familiar with the functionality. They were then asked to answer a questionnaire about the instrument controls. The questionnaire had three queries, listed below. They were answered on a 5-point Likert scale, where 1 is *Strongly disagree*, and 5 is *Strongly agree*.

1. *The temperature knob gives a feeling of control over the generated music.*
2. *The tempo knob gives a feeling of control over the generated music.*
3. *Changing the synth gives a feeling of control over the generated music.*

Part C: Model Evaluation

The third part of the study was an evaluation of the generative models. This evaluation was performed in order to understand how users perceive the output produced by the different generative models, to find out whether there are any perceived differences between the music generated by the models trained on different datasets, and whether the participants' perceptions match up with the expectations of the dataset qualities.

The participants spent a few minutes exploring each of the three generative models. Since studies show that serial positions of performances can have an impact on whether they are judged positively or negatively [87], and that they are also influenced by previous performances [88], the generative models were presented to the participants in a randomized order. After each model, they rated its performance with regards to the structural quality of the output and personal preference on a 5-point Likert scale:

1. *The generated music sounds good.*
2. *The generated music makes musical sense.*

Finally, once all three generative models were rated, the participants were asked to indicate which model was their favorite, and provide the reason behind their choice.

Part D: Interview

The final part of the session was a questionnaire with three open-ended queries listed below. The users were also asked to provide any additional comments they had that the previous questions did not cover.

1. *What did you like about instrument, and why?*
2. *What did you dislike about the instrument, and why?*
3. *Do you have any ideas for improvements?*

4.3.2 Data Analysis

The data resulting from parts B and C of a session were used for quantitative analysis. The answers to the question of which model was their favorite and why were also subjected to qualitative analysis. Due to the relatively small sample size of twelve participants, the main focus lies in the qualitative analysis.

Quantitive Analysis

The author employed statistical analysis for the Likert scale answers from session parts B and C to identify whether some of the instrument controls

or generative models were higher rated than the others, and to establish whether this difference was statistically significant. The analysis was done three times: once for each category of answers, which address the feeling of control given by the instrument controls, and whether the music generated by the models sounds good and makes musical sense.

First, a Kruskal–Wallis H test [89] was performed. The objective of the test is to see if it is possible to reject the null hypothesis, i.e., the medians of the ratings on the separate questions are equal. If they are equal, it indicates that the medians are from identical populations, and there is no evidence that any of the groups is different from the others. To reject the null hypothesis, the probability value (p-value) should be less than or equal to a significance level, usually chosen to be 0.05 [90]. If this is true, it shows that at least one group, or collection of ratings on a single question, dominates at least one other group. This test, however, does not indicate where does the dominance occur. Therefore, a Wilcoxon signed-rank test [91] was performed for pairwise comparisons on items where it was possible to reject the null hypothesis. This allowed identifying which pairs have significantly different median values.

Qualitative Analysis

To better understand the experience of interaction with the instrument, the interview questionnaires included four open-ended queries addressing the models the instrument was trained on and the instrument itself, in addition to a section for additional comments. Since this type of data, as well as the subjective experience of the users itself, cannot be easily quantifiable, the author employed thematic analysis to the data collected from the aforementioned questions.

Thematic analysis is a qualitative method used to identify patterns in the data and organize them in meaningful thematic units describing the results in detail and allowing for data interpretation. This method is used extensively in psychology and is argued to be "a foundational method for qualitative analysis" [92, p. 78]. Thematic analysis has also been used in NIME community [93] due to increasing attention to the importance of user- or performer evaluation.

The dataset for the analysis consists of answers to the queries provided by twelve participants, making a total of sixty answers. Of these, four respondents did not provide any content in the "additional comments" section; therefore, the total of answers considered in the analysis was fifty-six. The process of analysis followed the six phases described by Braun and Clarke [92, p. 87]: *familiarisation with data, coding data items, generating initial themes, reviewing themes, defining themes, and reporting*.

Since the interviews were done in English with non-native English speakers, some grammatical and lexical corrections of the text were made,

for instance, “componist” was translated to “composer.”

The coding phase was subdivided into three sub-phases: across-subject analysis of model preference, across-subject analysis of instrument feedback, and within-subject analysis of the entire dataset for all questions, including quantitative ratings. During the across-subject analysis of model preference, feedback on each of the three models was first analyzed separately to get insight into which qualities the participants found most “attractive” about it. It was followed by the analysis of feedback on all models together to extract the criteria the participants placed the greatest importance on in their evaluation of the performance. Across-subject analysis of instrument feedback was divided into three parts, considering first the answers to “what did you like about the instrument, and why?”, “what did you dislike about the instrument, and why?” and “do you have any ideas for improvements?” separately, and then all four questions, including additional comments, together. Finally, a within-subjects analysis of the entire dataset was conducted twice: first, for instrument feedback only, and then for both instrument feedback and model preference. This was done in an attempt to refine the initial themes by looking at the broader context of each data item.

Data items were coded according to multiple criteria, such as, which superordinate concept captured them best, what latent meanings could be grasped by the analyst, and the focus of the answer among others. For example, such items as “was more fun” and “It makes happy music” were coded as *enjoyment*, whereas “could change the feel of everything very easily” was coded four times as *agency* (the participant interacted actively), *control* (the participant liked that s/he could control it), *effort* (very easily), and *focus on musical output* (the feel of everything). These codes were then grouped according to apparent commonalities between them, and initial themes were generated. For example, codes like *irritation* and *enjoyment* were grouped as belonging to a more general theme of emotions. These themes were refined by re-examining all the data extracts belonging to a particular theme and within the context of each participant to finalize and define the thematic categories.

4.3.3 Results

Quantitative Analysis

Figure 4.5 shows the results from the questionnaire regarding instrument controls. All of the controls are rated fairly high with the majority of answers 4 or above, indicating that most participants felt that the controls gave them some degree of ownership over the generated music. The Kruskal–Wallis test gave a p-value of 0.37, which means that there is no statistically significant difference between the temperature, tempo and instrument sound selection with regards to how much control the

participants felt the different controls gave them.

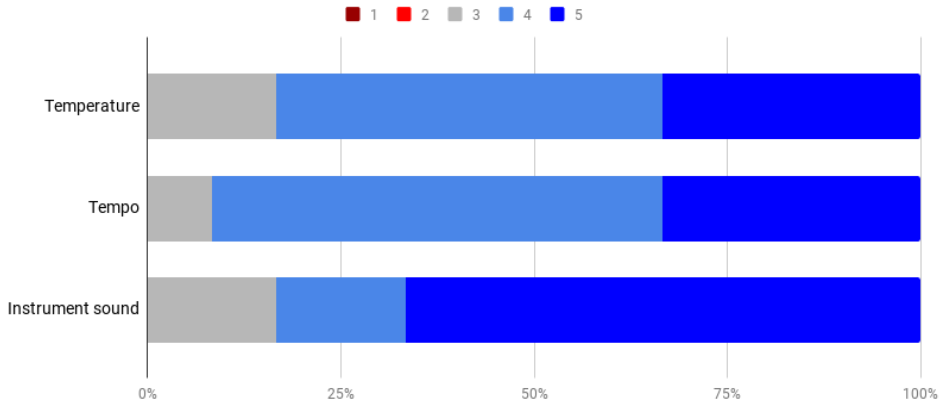


Figure 4.5: Results from the questionnaire on instrument controls. For each control knob, the questionnaire stated that the knob gives a feeling of control over the generated music. The participants rated the statements on a 5-point Likert scale, where 1 is *Strongly disagree* and 5 is *Strongly agree*. All ratings were relatively high, but the Kruskal–Wallis H test did not indicate any statistically significant differences between their ratings.

When the participants were asked if the generated music sounds good, most participants rated the output music trained on Bach chorales and Ryan's Mammoth Collection positively (Figure 4.6). The ratings provided on the output trained on the Final Fantasy 7 dataset, on the other hand, show slightly more varied results. Nevertheless, the differences between the ratings of the models are not substantial. Bach and Ryan's Mammoth seem to have the edge over Final Fantasy 7, having received more ratings of 4 and 5, but the Kruskal–Wallis test gives a p-value 0.11, which is not sufficiently low to conclude that there is a significant difference between their ratings.

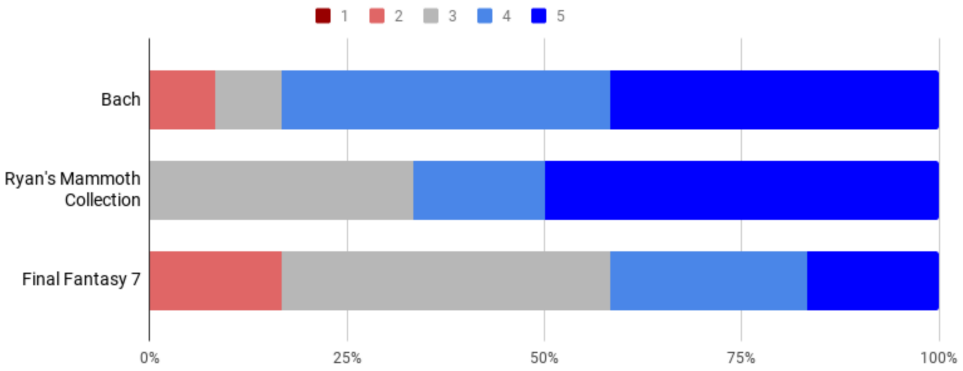


Figure 4.6: Responses to the statement *The generated music sounds good* for each generative model. The ratings were given on a 5-point Likert scale, where 1 is *Strongly disagree* and 5 is *Strongly agree*. Bach and Ryan's Mammoth Collection have somewhat higher ratings than Final Fantasy 7, but the differences are not large enough to be considered statistically significant.

The differences were more discernible when the participants were asked if the music generated by the three models made musical sense. Both Bach chorales and Ryan's Mammoth Collection were rated quite high, with almost all participants giving them a rating of 3 or above, while the same is true for only half of the ratings given to the Final Fantasy 7 model (Figure 4.7). The Kruskal-Wallis test gave a p-value of 0.03, which is low enough to assume that at least one group's median is significantly different from at least one other group's median. Bach and Ryan's Mammoth Collection received significantly higher ratings than Final Fantasy 7, but the two are not significantly different from each other, as seen in Figure 4.8, which shows the results of pairwise comparisons of the medians of the three populations using a Wilcoxon signed-rank test.

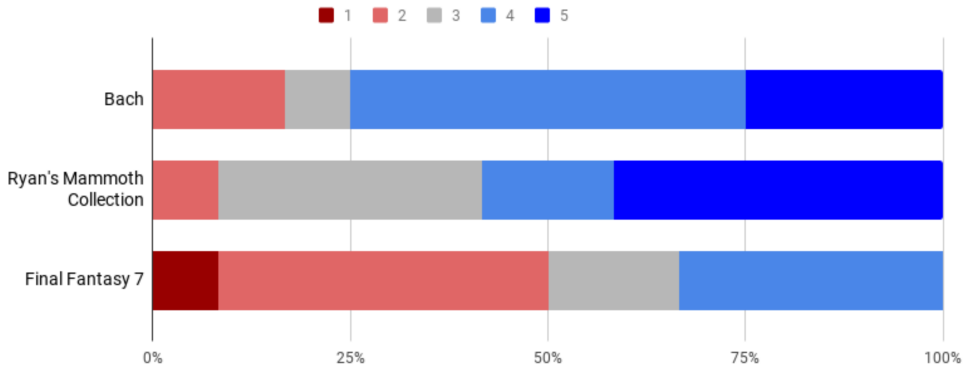


Figure 4.7: Responses to the statement *The generated music makes musical sense* for each generative model. The ratings were given on a 5-point Likert scale, where 1 is *Strongly disagree* and 5 is *Strongly agree*. The generative models trained on Bach chorales and Ryan's Mammoth Collection were rated significantly higher than Final Fantasy 7, but the differences between the two are insignificant.

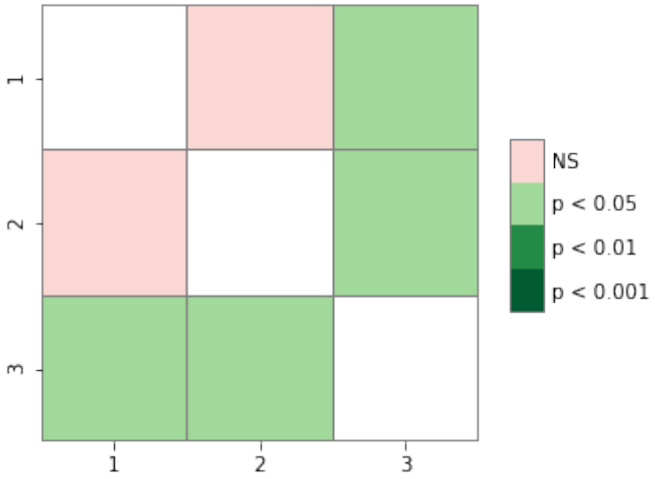


Figure 4.8: A heatmap of pairwise comparisons of the group medians for the ratings of the models on the statement *The generated music makes musical sense*. The axis labels indicate the different generative models: 1 - Bach chorales, 2 - Ryan's Mammoth Collection, and 3 - Final Fantasy 7. It is evident that the ratings of Bach chorales and Ryan's Mammoth Collection are significantly different from the ratings of Final Fantasy 7, but the former two are not different from each other.

Answers to the question on model preference clearly show that Ryan's Mammoth Collection is the winner, the preferred model of eight out of twelve participants (Figure 4.9). Three of the four remaining participants preferred the Bach chorales model, whereas Final Fantasy 7 was preferred by a single participant.

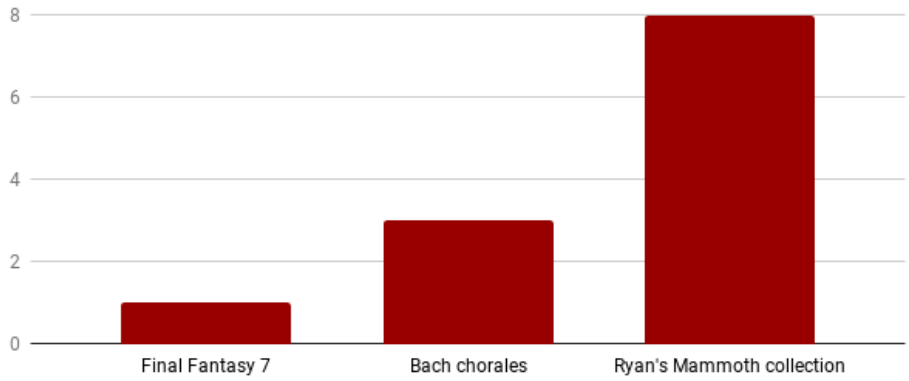


Figure 4.9: The model preferences indicated by the participants in the user study. The generative model trained on the Ryan's Mammoth Collection dataset is a clear winner chosen as the favorite by eight out of twelve participants.

Qualitative Analysis

Qualitative analysis resulted in three themes based on how people engaged with or perceived the prototype:

- The prototype as an instrument for live performance
- The prototype in between an instrument and a playback device
- The prototype as a playback device

Live performance. One participant saw the prototype as an instrument that could be used in live performances. S/he clearly felt in control of the music and seemed to be creatively engaged by it suggesting that "It would be nice to use it in a live setting together with other musicians and instruments {...}," and perceived it as original and providing the opportunity to improvise. The participant also appreciated that it did not require any experience with traditional instruments in order to do this. The focus on the live setting was also apparent in addressing what was disliked about the instrument by noting that "Changing models is not very smooth, particularly in a live setting it would be problematic." S/he also saw the generative models' tendency to become repetitive as a creative opportunity:

> "It was really cool to use the repeating notes as a build-up before change in melody, by slowly increasing tempo and then changing the temperature!"

This participant expressed a desire for more complexity in the produced output in terms of polyphony or possibility for a harmonious combination of different models, and mentions alternative positioning of the controls to give easier access to the most used controls. This participant is henceforth

referred to as the performer.

Instrument-playback device. Four participants seemed to perceive the prototype as something in between an instrument and a playback device. Oftentimes, their comments on design considerations included desires for the ability to record or repeat certain parts of the generated music, although it was not clear whether they saw that as important for a live setting, recording session, or private use. They also expressed a feeling of ownership over the produced music and enjoyed that this did not require experience in music composition. In the words of one such participant, "I liked the possibility of being my own composer, without knowing that much about composing music." Quite like the aforementioned performer, they generally seemed to feel in control of the music and wanted more complexity both in terms of model variety and the number of controls. They also saw the prototype as original, interesting, and were highly positive about it with remarks such as "So much fun, super innovative!" and "Tuning capabilities were various and thus gave a very flexible configuration of the song adapting it to everyone's taste." They also provided detailed feedback for improvements, such as better speaker quality, lowpass filter, and even "a visual representation of the sound wave."

Playback device. Six participants seemed to perceive the prototype more as a playback device. They could be characterized as not feeling so much in control over the resulting music even though they gave conflicting quantitative ratings of control. One particularly difficult to interpret participant gave mid-high quantitative ratings of control (4 for temperature, 4 for tempo, and 3 for synth), yet answered the question "What did you dislike about the instrument?" with "I didn't feel I had that much control over the result {...}". Participants in this group also seemed to be more focused on listening to the output rather than creating something with it, as in "{...} it was calming to listen to in a "mindful" kind of way." and comparing it to other music.

The majority of the so far mentioned participants perceived the prototype as interactive, for example, in answering what was liked about the instrument, one participant answered "The ability to rapidly change the controls and hear the result in real-time / very fast."

Finally, one participant did not fit any of the aforementioned groups because s/he did not seem to grasp the concept of the prototype in general, and the temperature control in particular, saying "I did not understand the temperature." Answering the question "What did you like about the instrument?" the participant answered "it was fun," which may be an indication that s/he perceived it as a toy, but feedback is rather limited to allow any further conclusions.

The reasons for why the participants preferred one or the other model were quite varied. In describing Ryan's Mammoth Collection model,

participants expressed enjoyment (particularly with the word "fun" or "makes happy music"), interest, coherency, and realness as main criteria for preference. Bach Chorales seemed to be more preferred due to personal tastes, with comments such as "Biased answer, into classical music" and "{...} reminds me of music I've heard before." The single participant who preferred Final Fantasy 7 said that "it sounded more epic," which is difficult to interpret.

4.4 Discussion

4.4.1 Feeling of Control and Interactivity

Although participants' ratings of control are mid-high, the same is not entirely reflected in the qualitative analysis. Sense of control seems to be most related to how they saw the prototype. Those who felt most in control were able to interact with it in a creative way and utilize the functions to produce output that they themselves rated as enjoyable, interesting, and original. Those who felt a lack of control, on the other hand, seemed less engaged with the instrument. Their interaction was rated as somewhat less positive, some even expressing high levels of irritation. There may be several reasons for the differing levels of perceived control. First, musical knowledge may have a strong impact. Although the user survey was anonymous and the author had no way to determine which participants provided which feedback, it is not implausible that the participants who were more engaged with the prototype might have some experience with music. The temperature knob presumes that the user has at least implicit understanding of relationships between notes in a melody and what changing them does to the output. A somewhat related reason may be the motivation to engage in creative musical activities in general. The performer, in particular, seemed to be motivated to use the opportunities provided by the instrument and used concrete examples from the session to describe the interaction. This shows that the instrument might be particularly relevant to people who wish to get an easy introduction to music-making. Finally, subjective taste in music has to be taken into account. The instrument was pre-trained on three datasets that have particular stylistic elements to them, which may not necessarily be liked by everyone. It is possible to train the instrument on datasets chosen by the user, effectively solving this problem. The code is available for anyone to use, and different datasets can be found online, for example, in the Music21 corpus.

Instrument Controls

The different control features provided by the instrument are high-level, essentially changing the general style and "feel" of the output. The quantitative analysis did not find any significant differences in the level of control provided by the different knobs. Somewhat similarly, qualitative analysis is not entirely straightforward in this regard either.

The participants who saw the prototype as an instrument or something in between an instrument and a playback device mentioned more often that they liked all the controls and appreciated the effects adjusting them had on the general output. The participants who seemed to perceive the prototype largely as a playback device only, however, appeared to be somewhat more preoccupied with the instrument sound knob. This may be because the instrument sound knob produces the most easily identifiable changes in the generated output and largely depends on the personal taste of a particular participant or user. The temperature knob, on the other hand, provides a high-level control of lower-level characteristic of the musical output, that is, the melodic pattern, or relationships between the notes, which creates changes in the output somewhat less obvious. As noted in the discussion of the feeling of control and interactivity, this may indicate that some basic understanding of musical knowledge is required. This may also be one of the reasons why the temperature was preferred by most at the NIME conference. Since NIME is a music technology conference, the audience was probably more motivated to engage with the creative opportunities the temperature control provides. It may also be that they have a higher tolerance for "unconventional" music than the average user study participant. Another explanation can be that the temperature is more appealing to someone who is interested in technology or machine learning, who might enjoy the fact that it is possible to control musical predictions made by a computer.

4.4.2 Musical Quality and Model Preference

The quantitative ratings of whether the music generated by the models makes musical sense showed that Ryan's Mammoth Collection and Bach chorales were rated significantly better than Final Fantasy 7. This is not unexpected since the Final Fantasy 7 dataset is less suited for monophonic music. Many of the instruments in the dataset scores have supportive roles and depend on other instruments to create pleasing harmonies. The Final Fantasy 7 dataset is likely better suited for a model architecture that allows generation of polyphonic music, or at least would require more preprocessing in order to make it work. The training loss of this model is also a little higher than the other two, indicating that it might have had more difficulties learning from the training examples. This is likely due to the dataset being less balanced. Ryan's Mammoth Collection and Bach chorales have a more similar musical structure in all training examples, while the sequences created from the Final Fantasy 7 dataset are more erratic. Another factor might be that the number of training sequences is much higher, and the relatively small number of parameters in the LSTM network is too low for this dataset, but high enough for Ryan's Mammoth Collection and Bach chorales.

Although the quantitative analysis showed that the differences between the ratings are not significant, Bach chorales and Ryan's Mammoth Collection datasets were more often rated with 4 or 5 than Final Fantasy 7

with regards to perceived musical quality. It is likely that the quality of the datasets had some impact on these results in the same way as for ratings addressing musical structure. However, what "sounds good" is a highly subjective measure and does not necessarily depend on the music-making sense. In fact, the ratings for whether a model sounds good and makes musical sense were sometimes quite inconsistent: the same participant may rate the musical quality as high, yet rate coherency as rather low. The current analysis does not lend itself to further interpretation of this discrepancy; however, it may well be that not all participants understood "music makes sense" in the same way.

4.4.3 Methodological Considerations

Although the user survey has provided insight into positive and negative aspects of the instrument, some methodological considerations must be taken into account. Although the survey was anonymous, the interviews were conducted by the designer of the prototype itself, making participants susceptible to demand characteristics. It might have been more beneficial to have an independent interviewer present the prototype. In addition, the statement formulations in the quantitative rating section may have influenced the answers provided to the open-ended queries. However, since it was desirable to acquire feedback in relation to the design requirements, it is not considered an issue in this thesis. In addition, the participants were told what dataset the training models were trained on. This may have influenced participants' perceptions of the generated music significantly, making them try to recognize familiar themes or compare the melody to the originals, instead of focusing on the instrument and the opportunities provided by it. Not revealing to the participants what datasets were used for training may have revealed more detailed feedback on the instrument and the musical quality. Finally, no information about the participants was collected. It would be beneficial to find out about the musical background, experience in information technologies, musical preferences, etc., in order to determine more easily who would benefit from the instrument most.

4.4.4 Design Considerations

There are two mentions of difficulty changing models, which can likely be explained by a loose connection that sometimes makes the rotary encoder non-responsive. One participant also mentioned that it was hard to select instrument sounds. This can possibly be because of the same reason as for changing models, but another explanation could be that the display does not show which instrument sound is currently active. Having the display show which instrument is active was one of the things participants mentioned most often, particularly those who saw the prototype as a playback device. Two of these participants also requested labels on the control knobs. It is possible that this would reduce the effort required to engage with the prototype.

Chapter 5

Conclusion

This thesis has introduced a novel self-contained intelligent instrument that uses a deep recurrent neural network to generate music and supports real-time user interaction. It demonstrates that real-time generation of music using recurrent neural networks is feasible on an embedded device with limited processing power.

There were two main goals of this thesis: to explore the potential of machine learning algorithms to generate music and investigate how high-level parameters can be used to shape the musical output, and to design and build a self-contained intelligent musical instrument on an embedded device.

The chosen machine learning algorithm for music generation is a character-level language model using LSTM recurrent neural networks. It is able to predict the next note in the musical sequence based on what it has already played, thus allowing for the generation of continuous melodies. Where most previous work on music generation is focused on offline generation of musical scores, this system can predict and play notes in real-time and is able to quickly respond to user manipulations, making it interactive. Users can tune five different parameters of the system: sampling temperature (diversity), tempo, volume, instrument sound selection, and generative model selection. Three generative models were trained on different datasets in order to have the option to choose from multiple musical styles.

Two instrument prototypes were made: an early version with basic functionality (sampling temperature and volume control), and an improved version with all the aforementioned instrument controls. They are both fully self-contained, with all the hardware necessary to create and control the music integrated into the encasement. The LSTM network runs on a Raspberry Pi single-board computer, and the control knobs are implemented using potentiometers and rotary incremental encoders.

A user study was performed to see the impact the different high-level parameter controls can have on a participant's perceived feeling of control over the musical output from the instrument, and to evaluate the three

trained generative models in terms of musical quality. The study included both Likert scale answers and open-ended queries, and it was evaluated both quantitatively and qualitatively. The study shows that participants felt a good amount of ownership over the music by using the control knobs on the prototype. The participants could be categorized into three groups according to how they seemingly perceived the prototype: as an instrument that can be used in a live setting, as something in between an instrument and a playback device, and as a playback device. The first category was able to become creatively engaged and utilize all the control knobs, and saw the potential for using the prototype in a live setting. The second group was also creatively engaged, but to a lesser extent than the former. However, both of these groups expressed a high feeling of control over the generated music and enjoyed that they did not have to be experienced in music composition to use the prototype. The last group was more passive and expressed lesser feel of control, although all ratings were mid-high in the quantitative analysis.

Although the prototype is aimed at beginners, it seems that some amount of understanding of music, as well as a desire to engage in the music-making process, is needed in order to get the most out of the temperature control. This was especially apparent by comparing the results from the user study with observations at the NIME conference. The NIME audience, which consists mostly of musicians and people interested in music technology, were clearly more focused on the temperature than participants in the user study.

Experience with the sampling temperature suggest that this interaction may help overcome some of the limitations of RNN music generation; in particular, the propensity for such models to become caught up in repetitive sequences. The fact that the RNN is made interactive may compensate for a model that is somewhat overtrained on a limited dataset. One of the participants in the user study was able to utilize this in a creative manner by letting the predictions become repetitive and use the repetitiveness as a build-up before increasing the temperature for more diverse predictions.

5.1 Future Work

5.1.1 Design Improvements

Hardware

The prototype was designed using a breadboard circuit which tended to have somewhat insecure connections (i.e., wires that would fall out) which could result in loss of function in the controllers if the instrument is handled carelessly. This could possibly be fixed by using some form of adhesive, but ideally, a printed circuit board (PCB) should be created. It was not prioritized due to time limitations but could be a beneficial upgrade to make the

instrument more robust.

Controls

Although it has not been tested, it should be possible to use the instrument in a studio setting. However, since songs are often recorded at a specific tempo, it can be challenging to find the exact BPM using the potentiometer tempo control since it makes the tempo scale continuous instead of discrete. Using a rotary incremental encoder instead, where each increment corresponds to a tempo increase of precisely one BPM, might be a better choice in this case. The BPM would also need to be shown on the display, especially since the rotary encoders can be turned 360 degrees, so the marker on the knob does not give any indication of the current tempo.

Generative Models

Hyperparameter optimization in order to improve the training and reduce overfitting of the LSTM models was not prioritized in this work, but is definitely a task that can be addressed in the future. The same is true for data pre-processing, especially for datasets like Final Fantasy 7, which would benefit from having a way of filtering out percussion tracks.

Two participants mentioned that they disliked the high-pitched notes that often appear at higher temperatures. This could be adjusted by limiting the range of possible pitches in the embedding layer and output layer, or by transposing high-pitched notes down one or two octaves during playback. However, the former would require that all datasets are within the same pitch range.

It would also be interesting to see whether it is possible to expand the model architecture so that it can play polyphonic music on the Raspberry Pi.

5.1.2 Additional Functionality

A possible upgrade to the instrument could be to have a MIDI output in addition to the speaker, allowing the instrument to act as a MIDI controller that can be connected to a desktop computer or laptop. In a studio setting, this would make it possible to use the instrument in combination with a DAW where the MIDI sequences can be sonified using a variety of instrument plugins. A MIDI interface where the user can connect a keyboard to play along with the prototype would also be an interesting function to investigate.

Appendices

Appendix A

NIME Paper

A Physical Intelligent Instrument using Recurrent Neural Networks

Torgrim R. Næss
Department of Informatics
University of Oslo, Norway
torgrirn@ifi.uio.no

Charles P. Martin
RITMO and Department of Informatics
University of Oslo, Norway
charlepm@ifi.uio.no

ABSTRACT

This paper describes a new intelligent interactive instrument, based on an embedded computing platform, where deep neural networks are applied to interactive music generation. Even though using neural networks for music composition is not uncommon, a lot of these models tend to not support any form of user interaction. We introduce a self-contained intelligent instrument using generative models, with support for real-time interaction where the user can adjust high-level parameters to modify the music generated by the instrument. We describe the technical details of our generative model and discuss the experience of using the system as part of musical performance.

Author Keywords

Embedded instruments, recurrent neural networks, generative models, interaction.

CCS Concepts

•Applied computing → Sound and music computing; •Computing methodologies → Neural networks; •Human-centered computing → *Interaction paradigms;*

1. INTRODUCTION

The use of machine learning algorithms to create musical compositions is a growing field of study. A lot of recent generative models applying deep neural networks, however, do not support direct human interaction, particularly in real-time during musical performance. Introducing ways for the user to manipulate the musical output will allow for such systems to be used in new kinds of intelligent interactive instruments, and to be explored through musical improvisation. Such instruments can be used by people with little or no previous experience to easily play around with musical ideas without the investment of time and money to learn a "real" instrument.

The main contribution of this research is a novel embedded device for interactively generating music with a recurrent neural network (RNN). This contrasts with many previous examples of music generation with neural networks that focus on offline, rather than interactive, generation of music, or require a powerful computer. Our device demonstrates that it is feasible to implement RNN-based music generation on a self-contained embedded platform, and explores manipulation of the continuous note sampling process as the main interactive function. This small device could be applied within many different musical scenarios and in setups with other instruments or equipment. This paper presents the design and implementation of an instrument prototype (Figure 1), and discusses the user experience along with some design considerations and ideas for future improvements.

Figure 1: A user interacting with our physical intelligent instrument: a device that continually generates and performs music using a recurrent neural network. The system is self-contained with controls for volume and sampling "temperature" as well as a built-in speaker for sonifying generated notes.

2. BACKGROUND

The use of artificial neural networks (ANNs) to create music and art has generated wide interest in recent years. ANNs can learn to transform any given picture to look like a painting created with different painting styles [9], and generate images that are practically indistinguishable from real photos [10]. ANNs can also be applied to synthesise musical audio waveform data [8], or translate music across musical instruments, genres, and styles [15].

Because neural networks can be trained to produce data learned from real-world examples, they can be good candidates for producing musical scores or performances without the need to program the rules of music theory manually. *Deep Artificial Composer* [5] can for example generate monophonic melodies resembling specified musical styles, and *Performance RNN* [17] can create polyphonic music with expressive timing and dynamics. Recently, interactive systems including ANN music generation have started to appear, e.g., *RoboJam* [13], a touchscreen music app that performs re-

NIME'19, June 3-6, 2019, Federal University of Rio Grande do Sul, Porto Alegre, Brazil.

sponses to short improvisations, and *Piano Genie* [7], which allows non-musicians to improvise on the piano. Neural nets can also be used to aid musicians in live performances, such as intelligent drum machines able to generate variants of rhythmic patterns provided as input by the user [19]. There is now potential to embed neural nets within smart musical instruments [18], or to create self-contained ANN music generators that could be used on stage or in the studio.

2.1 Artificial Neural Networks

ANNs were originally designed to imitate, in a simple way, the functionality of neurons in a real brain [4]. The basic building blocks are artificial neurons that take multiple input values, multiply them by respective weights and produce an output value. The input values can represent anything from pixels in an image to MIDI note values. Large numbers of these neurons are interconnected and arranged in a series of layers to form a network that can be trained for different applications such as pattern recognition and image classification. During training, the network is exposed to large numbers of examples with input values and expected output values. Based on the errors encountered in processing these examples, the network will adjust its weights to adapt to the training data.

2.1.1 Recurrent Neural Networks

Since music can be represented as a sequence of notes, the network must be able to predict this sequence based on both current and previous inputs; otherwise, the generated music will have no musical coherency. Recurrent neural networks (RNNs), are designed for the purpose of working with data sequences, and have been demonstrated to manage this task. The output of an RNN depends not only on its current inputs, but also on an internal state value that holds information from previous time steps, which allows it to learn to make decisions based on information it has seen in the past.

2.1.2 Long Short-Term Memory

Long short-term memory (LSTM) is a commonly used variant of RNNs that is able to capture much longer sequences than a regular simple RNN, which has a tendency to forget information when the sequences become longer. The LSTM achieves this by having an additional internal state called a "cell state" where information can be allowed to pass freely to later time steps, and by using a set of multiplicative gates to control the flow of information. There are three gates: an input gate, an output gate and a "forget gate". The input gate controls the information added to the cell state, making sure that irrelevant inputs do not pass through. Similarly, the output gate makes sure that irrelevant content is not passed along. The forget gate allows LSTM cells to reset when the content is no longer needed.

2.2 Music on Embedded Devices

Desktop computers and laptops are a well-established central component of digital musical instruments, but the use of single-board computers and embedded systems is becoming increasingly common. With the growing computing power and availability of single-board computers and microcontrollers, new platforms for creating digital musical instruments appear [14, 3]. Such platforms make it easy to prototype embedded instruments for performances and installations [16].

Self-contained, or embedded, instrument designs come with several advantages over the use of, for example, a laptop and simple microcontroller-based interface. The increased processing power of single-board computers allows more computationally intensive tasks to be performed natively than on a microcontroller, eliminating the need for external computers. Removing the use of general-purpose computers that are not dedicated to the instrument prototype can also increase longevity, as changes in other software might affect the functionality of the system [2]. The stability and portability of these systems suggest that they can be useful to artists who apply them within instrumental setups in live performance, or in their studios.

Figure 2: Close-up view of the physical intelligent instrument. The system has a built-in speaker and two knobs to control volume and sampling "temperature."

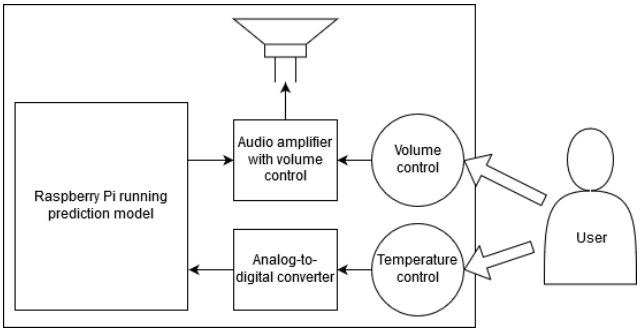


Figure 3: Diagram of the system. A generative music RNN model runs on a Raspberry Pi. The audio output goes through an amplifier with a potentiometer volume control and is played back on the built-in speaker. An analog-to-digital converter reads the voltage across a second potentiometer to control sampling temperature. The user can interact with the system by adjusting the two potentiometers.

3. DESIGN AND IMPLEMENTATION

Our physical intelligent instrument system, shown in Figure 2, consists of a box with a speaker and two knobs that the user can use to adjust certain parameters of musical predictions and playback. The system is designed to run music generation RNN models, and continually synthesise and sonify generated notes through an on-board speaker.

Figure 3 shows an overview of the system. The software runs on a Raspberry Pi Model 3 B+ single-board computer. An analog-to-digital converter (ADS1115) reads the voltage across a potentiometer to control the sampling temperature, and a 2.5 W mono audio amplifier (PAM8302), with another potentiometer for gain control, drives the speaker. The inside of the instrument is shown in Figure 4.

Figure 4: Hardware inside the enclosure. Hardware components consist of a Raspberry Pi, an analog-to-digital converter, two potentiometers, an audio amplifier and a speaker.

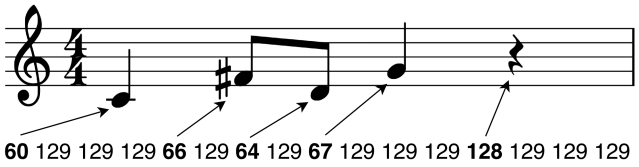


Figure 5: Notes are represented as integers with sixteenth note duration. 60, 64, 66 and 67 represent MIDI note numbers. 129 means no change, so the previous note will be held until a new note is played, or a value of 128 turns the note off.

3.1 Musical Representation

The system encodes music using sequences of integers in the range 0–129. 0–127 are pitches from the standard MIDI format, 128 tells the system to stop the note that was playing, and 129 represents no change. Each integer event has a duration of one sixteenth note. An example of the encoding is shown in Figure 5.

3.2 Network Architecture

The music generation RNN for the system is implemented in Python, using the Keras deep learning framework [4]. The RNN is in an auto-regression configuration, having been trained to predict the next in a sequence of notes. By returning the output of the RNN to the input, a continuous stream of musical notes can be generated.

The model consists of two LSTM layers with 256 cells each, an embedding layer on the input, and a dense layer with a softmax activation function on the output. The embedding layer transforms the input integer representation of notes into vectors of fixed size that the two LSTM layers can process hierarchically. The dense layer with softmax activation then projects the output from the LSTM layers back into probability distributions over possible note values. This style of neural network has been effectively used for creative sequence-learning tasks such as character-level text generation (CharRNN [11]), and music generation (MelodyRNN [1]).

3.3 Training

The training data is a set of 405 Bach chorales from the Music21 toolkit [6]. These chorales were split into four separate voices (soprano, alto, tenor and bass), and each voice was used as a single melody during training, giving a total of 1620 training examples of monophonic melodies. All melody lines were transposed to C major and A minor prior to training.

We trained the model for approximately 150 epochs with the Adam optimizer [12] and sparse categorical cross-entropy loss function, using a 90/10 training/validation split. Training was performed on an Nvidia GTX1070ti GPU. Due to the small size of the dataset, the model began to overfit before converging to a low loss value. It can be argued that this is not a big problem for this type of system, as the generated music should be perceived as pleasing to the human ear.

3.4 Sampling and Playback

When we sample from the model, it returns a vector of probability distributions over all possible note values. From this distribution, we can find the most probable next note based on previously predicted notes in the sequence. Two processes—one for sampling, and one for playback—run in parallel. The model is given a seed note as a starting point that it uses to predict the next notes in the sequence. After each prediction, the last note of the previous sequence is stored to be used as the seed for the next. This allows the model to play a continuous melody. The playback process sends notes over an internal MIDI connection for synthesis via the `Timidity++` software synthesiser and `FluidR3` soundfont. A shell script is used to ensure that the instrument processes are automatically started on boot. This, and the built-in speaker and controls, means that the instrument is completely self-contained and requires only USB power for performance.

3.5 User Interaction

After booting, the instrument will automatically start to play a continuous monophonic melody sampled from the LSTM network. By turning the two knobs on the instrument, the user can adjust two parameters of the system: volume and temperature. Temperature in this context is a hyperparameter that can be used to control the randomness of predictions by scaling outputs from the LSTM before applying softmax to calculate the probability distribution. A low temperature makes the model more conservative, making it less likely to sample from unlikely notes, while a higher temperature softens the probability distribution and allows the model to choose notes with a lower probability more often. Predictions with higher temperatures are more diverse, but can also have more mistakes. We use a temperature range of 0–10 in our model.

4. DISCUSSION

To give the feeling of real-time response from the instrument, we wanted to make sure the temperature adjustments begin to take effect as quickly as possible. Even though the system is fast enough to predict and play single notes, we found that predicting a sequence of two notes instead reduces the overhead without making any noticeable change in the response time from the temperature adjustment controls. This is not a big concern at this time, but might become more important when expanding the instrument with more functionality in the future.

The most important interaction in this context is the temperature control, which has a big impact on the generated melodies. At a temperature of 1, we can hear that the LSTM network has learned the structure and important musical elements of Bach chorales. When we increase the temperature to mid-range, the music begins to sound more like

experimental jazz, while still retaining some of the chorale elements. At higher temperatures, the music sounds like completely random notes, especially at max temperature where there is almost no coherency or structure. At the lowest temperature, the music tends to get stuck at a single note or pattern very quickly. We would argue that having the option to control the diversity of the generated melody results in a more enjoyable and rewarding user experience. There is another advantage of having a way to control the sampling temperature. When generating longer sequences, an LSTM network will often converge to a fixed state and become very repetitive. If this happens, the user can increase the temperature to make sure the model will include some more unlikely notes.

In our experience, the instrument is quite fun to use, and it grants the feeling of having some control over the music, even though it is generated by the LSTM model. However, more ways of interaction could be added in order to make this prototype a complete instrument, as it can become a little monotonous after a while. Possible extensions could be tempo control, choice of software synthesisers, or a MIDI interface where the user can connect a keyboard to play along with the instrument. It could also be useful to implement a way of choosing between multiple models trained on different data sets. If the users wants to train and test their own models, being able to switch between them would result in a more versatile instrument.

Experience with this system so far suggests that music generation RNNs could be a useful tool in music-making for both unskilled and expert users. Our system allows the sampling process to be explored in real-time during improvisation. As our system is self-contained, it can easily be integrated with other music-making devices, such as hardware synthesisers, or commercial music controllers.

5. CONCLUSIONS

In this paper, we have introduced a self-contained, physical, intelligent instrument that uses a deep recurrent neural network to generate music and supports real-time user interaction. Our work so far has demonstrated that real-time generation of music using an RNN is feasible on an embedded instrument using a Raspberry Pi. This system allows sampling diversity, or temperature, of the neural network model to be explored in real-time performance. Our initial explorations with the system suggest that this interaction, albeit simple, may help over some of the limitations of RNN music generation; in particular, the propensity for such models to become caught up in repetitive sequences. The fact that the RNN is made interactive may compensate for a model that is somewhat overtrained on a limited dataset. In future work we seek to understand how this system could form part of everyday music-making setups, and what kind of music it could afford or suggest. We are currently experimenting with an enhanced system that allows users to switch between different trained RNN models, between different software synthesisers, and to change tempo. This allows music from different styles or idioms to be generated. Code and schematics for both instrument versions is openly accessible on GitHub.[1]

6. ACKNOWLEDGMENTS

This work is supported by The Research Council of Norway as part of the Engineering Predictability with Embodied Cognition (EPEC) project #240862, the Collaboration on Intelligent Machines (COINMAC) project #261645, and the Centres of Excellence scheme, project #262762.

[1] https://github.com/edrukar/intelligent_instrument

7. REFERENCES

[1] D. Abolafia. A recurrent neural network music generation tutorial. [Magenta Project Blog Post], 2016.

[2] E. Berdahl. How to make embedded acoustic instruments. In *Proc. NIME '14*, pages 140–143, 2014.

[3] E. Berdahl and W. Ju. Satellite CCRMA: A musical interaction and sound synthesis platform. In *Proc. NIME '11*, pages 173–178, 2011.

[4] F. Chollet. *Deep learning with Python*. Manning Publications Co, 2018.

[5] F. Colombo, A. Seeholzer, and W. Gerstner. Deep artificial composer: A creative neural network model for automated melody generation. In *Computational Intelligence in Music, Sound, Art and Design*, pages 81–96. Springer, 2017.

[6] M. S. Cuthbert and C. Ariza. Music21: A toolkit for computer-aided musicology and symbolic music data. In *ISMIR*, pages 637–642. International Society for Music Information Retrieval, 2010.

[7] C. Donahue, I. Simon, and S. Dieleman. Piano genie. In *Proc. IUI*, 2019. doi:10.1145/3301275.3302288.

[8] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi. Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders. In *Proc. ICML*, 2017.

[9] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proc. CVPR*, pages 2414–2423, 2016.

[10] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra. Draw: A recurrent neural network for image generation. In *Proc. ICML*, volume 37, pages 1462–1471, 2015.

[11] A. Karpathy. The unreasonable effectiveness of recurrent neural networks. Published on Andrej Karpathy's blog, May 2015.

[12] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR 2015*, 2015.

[13] C. P. Martin and J. Torresen. RoboJam: A musical mixture density network for collaborative touchscreen interaction. In *Computational Intelligence in Music, Sound, Art and Design: International Conference, EvoMUSART*, 2018. doi:10.1007/978-3-319-77583-8_11.

[14] A. McPherson and V. Zappi. An environment for submillisecond-latency audio and sensor processing on beaglebone black. In *Audio Engineering Society Convention 138*. Audio Engineering Society, 2015.

[15] N. Mor, L. Wolf, A. Polyak, and Y. Taigman. A universal music translation network. *ArXiV ePrint*, 2018. arXiv:1805.07848.

[16] V. E. G. Sanchez, A. Zelechowska, C. P. Martin, V. Johnson, K. A. V. Bjerkestrand, and A. R. Jensenius. Bela-based augmented acoustic guitars for inverse sonic microinteraction. In *Proc. NIME '18*, page 324–327, 2018.

[17] I. Simon and S. Oore. Performance rnn: Generating music with expressive timing and dynamics. [Magenta Project Blog Post], June 2017.

[18] L. Turchet. Smart musical instruments: Vision, design principles, and future directions. *IEEE Access*, 7:8944–8963, 2019.

[19] R. Vogl and P. Knees. An intelligent drum machine for electronic dance music production and performance. In *Proc. NIME '17*, pages 251–256, 2017.

Bibliography

[1] I. Poupyrev, M. J. Lyons, S. Fels and T. Blaine (Bean), 'New interfaces for musical expression', in *CHI '01 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '01, Seattle, Washington: ACM, 2001, pp. 491–492. DOI: 10.1145/634067.634348.

[2] S. Dubnov, G. Assayag, O. Lartillot and G. Bejerano, 'Using machine-learning methods for musical style modeling', *Computer*, vol. 36, no. 10, pp. 73–80, Oct. 2003. DOI: 10.1109/MC.2003.1236474.

[3] B. Caramiaux and A. Tanaka, 'Machine learning of musical gestures', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Daejeon, Republic of Korea, May 2013, pp. 513–518.

[4] *Magenta*, Magenta. [Online]. Available: https://magenta.tensorflow.org/ (visited on 30/07/2018).

[5] S. Oore, I. Simon, S. Dieleman, D. Eck and K. Simonyan, 'This time with feeling: Learning expressive musical performance', *Neural Computing and Applications*, Nov. 2018. DOI: 10.1007/s00521-018-3758-9.

[6] *NIME – The International Conference on New Interfaces for Musical Expression*. [Online]. Available: https://www.nime.org/ (visited on 31/05/2019).

[7] P. R. Cook, 'Principles for designing computer music controllers', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Seattle, WA, 2001, pp. 3–6.

[8] T. Mitchell and I. Heap, 'Soundgrasp : A gestural interface for the performance of live music', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Oslo, Norway, 2011, pp. 465–468.

[9] O. Bown and S. Ferguson, 'A musical game of bowls using the diads', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, vol. 16, Brisbane, Australia, 2016, pp. 371–372.

[10] V. E. Gonzalez Sanchez, C. P. Martin, A. Zelechowska, K. A. V. Bjerkestrand, V. Johnson and A. R. Jensenius, 'Bela-based augmented acoustic guitars for sonic microinteraction', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Blacksburg, Virginia, USA, Jun. 2018, pp. 324–327.

[11] G. Beyer and M. Meier, 'Music interfaces for novice users : Composing music on a public display with hand gestures', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Oslo, Norway, 2011, pp. 507–510.

[12] M. Hölzl, G. Denker, M. Meier and M. Wirsing, 'Constraint-muse: A soft-constraint based system for music therapy', in *Algebra and Coalgebra in Computer Science*, Berlin, Heidelberg, 2009, pp. 423–432.

[13] M. Luhtala, T. Kymäläinen and J. Plomp, 'Designing a music performance space for persons with intellectual learning disabilities', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Oslo, Norway, 2011, pp. 429–432.

[14] A. McPherson and V. Zappi, 'An environment for submillisecond-latency audio and sensor processing on beaglebone black', in *Audio Engineering Society Convention 138*, Jan. 2015, pp. 965–971.

[15] *BeagleBoard.org - black*. [Online]. Available: https://beagleboard.org/black (visited on 25/03/2019).

[16] G. Moro, A. Bin, R. H. Jack, C. Heinrichs and A. P. McPherson, 'Making high-performance embedded instruments with bela and pure data',

[17] V. E. Gonzalez Sanchez, C. P. Martin, A. Zelechowska, K. A. V. Bjerkestrand, V. Johnson and A. R. Jensenius, *Sverm-Resonans at Ultima - RITMO Centre for Interdisciplinary Studies in Rhythm, Time and Motion*. [Online]. Available: https://www.uio.no/ritmo/english/projects/all/sverm/events/2017/ultima/index.html (visited on 31/07/2019).

[18] E. Berdahl and W. Ju, 'Satellite ccrma: A musical interaction and sound synthesis platform', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Oslo, Norway, 2011, pp. 173–178.

[19] E. Berdahl, S. Salazar and M. Borins, 'Embedded networking and hardware-accelerated graphics with satellite ccrma', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Daejeon, Republic of Korea, May 2013, pp. 325–330.

[20] E. Berdahl and C. Chafe, 'Autonomous new media artefacts (autonma)', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Oslo, Norway, 2011, pp. 322–323.

[21] E. Berdahl, 'How to make embedded acoustic instruments', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, London, United Kingdom, 2014, pp. 140–143.

[22] A. Pajankar, 'Introduction to single board computers and raspberry pi', in *Raspberry Pi Image Processing Programming: Develop Real-Life Examples with Python, Pillow, and SciPy*. Berkeley, CA: Apress, 2017, pp. 1–24, ISBN: 978-1-4842-2731-2.

[23] *Raspberry Pi — Teach, Learn, and Make with Raspberry Pi*. [Online]. Available: https://www.raspberrypi.org/ (visited on 25/03/2019).

[24] *Tinker Board | Single Board Computer | ASUS Global*. [Online]. Available: https://www.asus.com/Single-Board-Computer/Tinker-Board/specifications/ (visited on 25/03/2019).

[25] *Embedded Systems Developer Kits, Modules, & SDKs | NVIDIA Jetson*. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines / embedded - systems - dev - kits - modules/ (visited on 25/03/2019).

[26] J. Engel, C. Resnick, A. Roberts, S. Dieleman, M. Norouzi, D. Eck and K. Simonyan, 'Neural audio synthesis of musical notes with wavenet autoencoders', in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, Sydney, NSW, Australia, 2017, pp. 1068–1077.

[27] N. Mor, L. Wolf, A. Polyak and Y. Taigman, 'A universal music translation network', *ArXiV ePrint*, 2018. arXiv: 1805.07848.

[28] M. C. MOZER, 'Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing', *Connection Science*, vol. 6, no. 2-3, pp. 247–280, 1994. DOI: 10.1080/09540099408915726.

[29] S. Hochreiter and J. Schmidhuber, 'Long short-term memory', *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[30] D. Eck and J. Schmidhuber, 'Finding temporal structure in music: Blues improvisation with lstm recurrent networks', in *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, Sep. 2002, pp. 747–756. DOI: 10.1109/NNSP.2002.1030094.

[31] B. L. Sturm, J. F. Santos, O. Ben-Tal and I. Korshunova, 'Music transcription modelling and composition using deep learning', in *Proceedings of the 1st Conference on Computer Simulation of Musical Creativity*, 2016.

[32] F. Colombo, A. Seeholzer and W. Gerstner, 'Deep artificial composer: A creative neural network model for automated melody generation', in *Computational Intelligence in Music, Sound, Art and Design*, Cham, 2017, pp. 81–96.

[33] H. Lim, S. Rhyu and K. Lee, 'Chord generation from symbolic melody using BLSTM networks', in *Proc. ISMIR*, 2017. arXiv: 1712.01011.

[34] A. Faitas, S. E. Baumann, T. R. Næss, J. Torresen and C. P. Martin, 'Generating convincing harmony parts with simple long short-term memory networks', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Porto Alegre, Brazil, Jun. 2019, pp. 325–330.

[35] G. Grimmett and D. Stirzaker, *Probability and random processes*, ser. Oxford science publications. Clarendon Press, 1985, ISBN: 9780198531852.

[36] G. Nierhaus, *Algorithmic Composition: Paradigms of Automated Music Generation*, eng. Vienna: Springer Vienna, 2009, ISBN: 9783211755396.

[37] M. Allan and C. K. I. Williams, 'Harmonising chorales by probabilistic inference', in *Advances in Neural Information Processing Systems*, vol. 17, 2005.

[38] I. Simon, D. Morris and S. Basu, 'Mysong: Automatic accompaniment generation for vocal melodies', in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Florence, Italy, 2008, pp. 725–734. DOI: 10.1145/1357054.1357169.

[39] P. M. Todd and G. Loy, 'A connectionist approach to algorithmic composition', in *Music and Connectionism*. MITP, 2003, ISBN: 9780262285032.

[40] A. Graves, 'Generating sequences with recurrent neural networks', *ArXiv*, vol. abs/1308.0850, 2013.

[41] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel and D. Eck, 'Enabling factorized piano music modeling and generation with the MAESTRO dataset', in *International Conference on Learning Representations*, 2019.

[42] L. Turchet, 'Smart musical instruments: Vision, design principles, and future directions', *IEEE Access*, vol. 7, pp. 8944–8963, 2019.

[43] C. Donahue, I. Simon and S. Dieleman, 'Piano genie', in *Proceedings of the 24th International Conference on Intelligent User Interfaces*, Marina del Ray, California, 2019, pp. 160–164. DOI: 10.1145/3301275.3302288.

[44] *NSynth Super*. [Online]. Available: https://nsynthsuper.withgoogle.com/ (visited on 25/06/2019).

[45] R. Vogl and P. Knees, 'An intelligent drum machine for electronic dance music production and performance', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Copenhagen, Denmark, 2017, pp. 251–256.

[46] F. Pachet, 'The continuator: Musical interaction with style', *Journal of New Music Research*, vol. 32, pp. 333–341, Aug. 2010. DOI: 10.1076/jnmr.32.3.333.16861.

[47] *Play a duet with a computer, through machine learning*, Google, 16th Feb. 2017. [Online]. Available: https://www.blog.google/technology/ai/play-duet-computer-through-machine-learning/ (visited on 01/08/2018).

[48] C. P. Martin, K. O. Ellefsen and J. Torresen, 'Deep models for ensemble touch-screen improvisation', in *Proceedings of the 12th International Audio Mostly Conference on Augmented and Participatory Sound and Music Experiences*, London, United Kingdom, Aug. 2017. DOI: 10.1145/3123514.3123556.

[49] C. P. Martin and J. Torresen, 'Robojam: A musical mixture density network for collaborative touchscreen interaction', in *Computational Intelligence in Music, Sound, Art and Design*, 2018, pp. 161–176. DOI: 10.1007/978-3-319-77583-8_11.

[50] F. Chollet, *Deep Learning with Python*. Manning, Nov. 2017, ISBN: 9781617294433.

[51] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[52] G. Chen, 'A gentle tutorial of recurrent neural network with error backpropagation', *CoRR*, vol. abs/1610.02583, 2016. arXiv: 1610.02583. [Online]. Available: http://arxiv.org/abs/1610.02583.

[53] L. Verwimp, J. Pelemans, H. V. hamme and P. Wambacq, 'Character-word lstm language models', in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, 2017, pp. 417–427.

[54] P. P. Barman and A. Boruah, 'A rnn based approach for next word prediction in assamese phonetic transcription', eng, *Procedia Computer Science*, vol. 143, pp. 117–123, 2018.

[55] I. Sutskever, J. Martens and G. Hinton, 'Generating text with recurrent neural networks', in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, Bellevue, Washington, USA, 2011, pp. 1017–1024.

[56] T. Mikolov, I. Sutskever, A. Deoras, L. Hai Son, S. Kombrink and J. Cernock, 'Subword language modeling with neural networks', Jun. 2012.

[57] D. Abolafia, *A recurrent neural network music generation tutorial*, [Magenta Project Blog Post], 2016. [Online]. Available: https://magenta.tensorflow.org/2016/06/10/recurrent-neural-network-generation-tutorial.

[58] K. Choi, G. Fazekas and M. B. Sandler, 'Text-based lstm networks for automatic music composition', in *Proceedings of the 1st Conference on Computer Simulation of Musical Creativity*, 2016.

[59] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, 'Learning phrase representations using RNN encoder–decoder for statistical machine translation', in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, Oct. 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.

[60] I. Sutskever, O. Vinyals and Q. V. Le, 'Sequence to sequence learning with neural networks', in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, Montreal, Canada, 2014, pp. 3104–3112.

[61] F. Chollet, *A ten-minute introduction to sequence-to-sequence learning in Keras*. [Online]. Available: https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html (visited on 31/07/2019).

[62] T. R. Næss and C. P. Martin, 'A physical intelligent instrument using recurrent neural networks', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Porto Alegre, Brazil, Jun. 2019, pp. 79–82.

[63] D. Self, *Small signal audio design*. amsterdam: Elsevier, 2010, ISBN: 9780240521770.

[64] *TensorFlow 1.9 Officially Supports the Raspberry Pi - Medium*. [Online]. Available: https://medium.com/tensorflow/tensorflow-1-9-officially-supports-the-raspberry-pi-b91669b0aa0 (visited on 26/06/2019).

[65] M. Abadi *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: http://tensorflow.org/.

[66] *Potentiometer*. [Online]. Available: http://www.resistorguide.com/potentiometer/ (visited on 11/07/2019).

[67] *ADS1115 16-Bit ADC - 4 Channel with Programmable Gain Amplifier*. [Online]. Available: https://www.adafruit.com/product/1085 (visited on 26/06/2019).

[68] *Adafruit Mono 2.5W Class D Audio Amplifier - PAM8302*. [Online]. Available: https://www.adafruit.com/product/2130 (visited on 21/06/2019).

[69] *GM 1 Sound Set*. [Online]. Available: https://www.midi.org/specifications-old/item/gm-level-1-sound-set (visited on 26/06/2019).

[70] *Introduction to Incremental Encoders*. [Online]. Available: http://www.sensoray.com/support/appnotes/encoders.htm (visited on 26/06/2019).

[71] *Monochrome 0.96 128x64 OLED graphic display*. [Online]. Available: https://www.adafruit.com/product/326 (visited on 21/06/2019).

[72] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[73] A. Graves, A.-r. Mohamed and G. E. Hinton, 'Speech recognition with deep recurrent neural networks', *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649, 2013.

[74] M. S. Cuthbert and C. Ariza, *Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data*, 2010.

[75] *The Midi Shrine peps - Game Music MIDI files*. [Online]. Available: http://www.midishrine.com/ (visited on 25/03/2019).

[76] P. Sky, *Ryan's Mammoth Collection of Fiddle Tunes*. Mel Bay Publications, Inc, Sep. 1995, ISBN: 0-7866-0300-3.

[77] C. Martin, *Cpmpercussion/creative-prediction v1.0 [Git Repository]*, Nov. 2018. DOI: 10.5281/zenodo.1494040.

[78] G. Mazzola, 'Symmetries and Morphisms', in *The Topos of Music I: Theory: Geometric Logic, Classification, Harmony, Counterpoint, Motives, Rhythm*, Cham: Springer International Publishing, 2017, pp. 113–144, ISBN: 978-3-0348-8141-8.

[79] D. Scherer, A. Müller and S. Behnke, 'Evaluation of pooling operations in convolutional architectures for object recognition', in *Artificial Neural Networks – ICANN 2010*, Berlin, Heidelberg, 2010, pp. 92–101.

[80] J. Liu, A. Shahroudy, D. Xu and G. Wang, 'Spatio-temporal lstm with trust gates for 3d human action recognition', in *Computer Vision – ECCV 2016*, Cham, 2016, pp. 816–833.

[81] D. P. Kingma and J. Ba, 'Adam: A method for stochastic optimization', in *Proc. ICLR*, 2015. arXiv: 1412.6980.

[82] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, 'Dropout: A simple way to prevent neural networks from overfitting', *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[83] A. Karpathy, *The unreasonable effectiveness of recurrent neural networks*, Published on Andrej Karpathy's blog, May 2015. [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[84] J. Barbosa, J. Malloch, M. M. Wanderley and S. Huot, 'What does "Evaluation" mean for the NIME community?', in *NIME 2015 - 15th International Conference on New Interfaces for Musical Expression*, May 2015, pp. 156–161.

[85] S. O'Modhrain, 'A framework for the evaluation of digital musical instruments', *Computer Music Journal*, vol. 35, pp. 28–42, Mar. 2011. DOI: 10.1162/COMJ_a_00038.

[86] L.-C. Yang and A. Lerch, 'On the evaluation of generative models in music', *Neural Computing and Applications*, Nov. 2018. DOI: 10.1007/s00521-018-3849-7.

[87] W. Bruine de Bruin, 'Save the last dance for me: Unwanted serial position effects injury evaluations', *Acta psychologica*, vol. 118, pp. 245–60, Apr. 2005. DOI: 10.1016/j.actpsy.2004.08.005.

[88] L. Damisch, T. Mussweiler and H. Plessner, 'Olympic medals as fruits of comparison? assimilation and contrast in sequential performance judgments', *Journal of experimental psychology. Applied*, vol. 12, pp. 166–78, Oct. 2006. DOI: 10.1037/1076-898X.12.3.166.

[89] *Kruskal-Wallis H Test in SPSS Statistics*. [Online]. Available: https://statistics.laerd.com/spss-tutorials/kruskal-wallis-h-test-using-spss-statistics.php (visited on 13/07/2019).

[90] *Alpha Level (Significance Level): What is it?* [Online]. Available: https://www.statisticshowto.datasciencecentral.com/what-is-an-alpha-level/ (visited on 28/07/2019).

[91] *Wilcoxon Signed Rank Test: Definition, How to Run*. [Online]. Available: https://www.statisticshowto.datasciencecentral.com/wilcoxon-signed-rank-test/ (visited on 13/07/2019).

[92] V. Braun and V. Clarke, 'Using thematic analysis in psychology', *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006. DOI: 10.1191/1478088706qp063oa.

[93] A. Tanaka, A. Parkinson, Z. Settel and K. Tahiroglu, 'A survey and thematic analysis approach as input to the design of mobile music guis', in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Ann Arbor, Michigan, 2012.