MUSIC REPRESENTING CORPUS VIRTUAL: AN OPEN SOURCED LIBRARY FOR EXPLORATIVE MUSIC GENERATION, SOUND DESIGN, AND INSTRUMENT CREATION WITH ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

A PREPRINT

Christopher Johann Clarke
In Fulfillment of The National Arts Council Creation Grant
National Arts Council, Singapore
chris.johann.clarke@gmail.com

May 2, 2023

ABSTRACT

Music Representing Corpus Virtual (MRCV) is an open source software suite designed to explore the capabilities of Artificial Intelligence (AI) and Machine Learning (ML) in Music Generation, Sound Design, and Virtual Instrument Creation (MGSDIC). The software is accessible to users of varying levels of experience, with an emphasis on providing an explorative approach to MGSDIC. The main aim of MRCV is to facilitate creativity, allowing users to customize input datasets for training the neural networks, and offering a range of options for each neural network (thoroughly documented in the Github Wiki). The software suite is designed to be accessible to musicians, audio professionals, sound designers, and composers, regardless of their prior experience in AI or ML. The documentation is prepared in such a way as to abstract technical details, thereby making it easy to understand. The software is open source, meaning users can contribute to its development, and the community can collectively benefit from the insights and experience of other users.

Keywords Artificial Intelligence · Machine Learning · Education · Music Generation · Sound Design · Instrument Creation · Software · Audio DSP

1 Introduction

Artificial Intelligence (AI) and Machine Learning (ML) have been areas of active research for several decades, with roots that can be traced back to the 1940s and 1950s [Wilson, 2011]. The idea of creating machines capable of emulating human thought and behavior has been a topic of interest since the early days of computing, and the development of AI and ML has been driven by various factors, including improvements in hardware, the availability of larger datasets, and advancements in algorithms and techniques [Lu et al., 2018].

In recent years, ML has experienced a significant surge of interest due to advancements in deep learning, reinforcement learning, and other techniques. These breakthroughs have led to significant progress in areas such as image and speech recognition, natural language processing, and predictive modeling, among others [Howard, 2019]. The adoption of ML has rapidly spread across numerous industries, including music, where ML is now being used for music generation [Kaliakatsos-Papakostas et al., 2020], sound design [Miranda, 1995], and virtual instrument creation [Tahiroğlu et al., 2020].

Algorithmic music composition has a long history, with early experiments dating back to the 1950s and 1960s. The rise of digital audio workstations (DAWs) in the 1980s and 1990s [Jackson and Jackson, 2015] facilitated the creation of more sophisticated music software, which led to the development of algorithmic music composition techniques. These

techniques involve the use of computer algorithms to generate musical material, either independently or in collaboration with human composers [Alpern, 1995].

With the rise of ML techniques in music, the field has experienced an upsurge in interest in using these techniques for music generation, sound design, and virtual instrument creation. Researchers and musicians are exploring the use of ML in music in innovative ways, ranging from the application of supervised and unsupervised learning algorithms to the development of deep neural networks for music analysis [Miranda, 2013] and synthesis [Roads, 1985]. These new techniques have opened up exciting possibilities for musical creativity and exploration, offering new tools for musicians and composers to experiment with and push the boundaries of what is possible in music.

This paper will take a descriptive view on the software library developed, explaining each component part of the library. This serves as a means for further development. As a companion aid to the documentation on the Github, this will highlight the formal structure of the library, and the reasoning behind the design choices made. The paper will also discuss the future development of the library, and the potential for further research in the field of AI and ML in music. The potential implications of this software are outside of the scope of the discussion herein.

2 Descriptive View of the Software Library

2.1 Preface

In order to provide formal description of the various neural networks, this subsection is dedicated to expressing some of the language and notation used in the sections ahead. Firstly, a dense neural network can be described thusly:

$$\hat{y} = \mathbb{M}(x) = \sigma(Wx + b) \circ \cdots \circ \sigma(Wx_0 + b) \tag{1}$$

Where $\mathbb{M}$ represents the model, which is composed ($\circ$ denotes function composition) of multiple layers. Each layer $\sigma(Wx + b)$ has an activation function σ, a weight value W, an input x, and a bias value b. x_0 is defined as the first input to the model, and $\hat{y}$ is the output of the model.

In most cases, the input x comes from feature extraction. In the case of audio, an example of this is an FFT of the signal at a given time frame. This can be represented as such, borrowing from equation 1:

$$\hat{y} = \mathbb{M}(\mathcal{F}(x)) = \sigma(W(x) + b) \circ \cdots \circ \sigma(W\mathcal{F}(x_0) + b) \tag{2}$$

Where $\mathcal{F}$ is the feature extraction method used. From Equation 2, this feature extraction method is only applicable to the first input of the network.

In order to efficiently notate a model's parameters, such as layer width (number of neurons per layer) and layer count (number of layers), the following shorthand notation is used:

$$\mathbb{M} = \left[\sum_{0}^{(L,\mathbf{p})} \right] \tag{3}$$

Where L is the layer count, and $\mathbf{p}$ is the layer width. For example, a model with 3 layers, with 2 neurons per layer, can be notated as such:

$$\mathbb{M}_{layers=3,width=2} = \left[\sum_{0}^{(3,2)} \right] \tag{4}$$

This notation is used throughout the paper to describe the various models and submodels used.

2.2 Motivations in Explorative Searching

Generally, this software library makes use of the trained latent representations of the dataset as a sort of pseudo-random generator. However, this pseudo-randomness maintains some sort of congruence to the training dataset. A latent representation in a neural network refers to the hidden layers that capture a compressed, abstract representation of the input data. The goal of this layer is to extract meaningful and relevant features from the input to the layer and transform these input data into a differently compact or expressive compressed form.

During the training process, the neural network learns to map the input data onto this latent representation by adjusting the weights of the connections between the layers. This process is known as feature extraction and is critical to the success of many machine learning tasks, such as image classification, natural language processing, and music generation.

The term "latent" refers to the fact that the representation is not directly observable in the input data but is inferred from the patterns and relationships within the data. This compressed representation can be thought of as a high-dimensional abstraction of the input data that captures the essential characteristics of the data, while discarding irrelevant details.

Once the neural network has learned a good latent representation of the input data, it can be used for a variety of tasks, such as generating new data that is similar to the input data or classifying the input data into different categories. Overall, latent representations are a powerful tool for neural networks and are a key factor in their ability to learn complex relationships within data.

However, when used in this case, the latent representations serve as a means of mixing multiple characteristics into one. For example, the latent representation of a saxophone note can be mixed with the latent representation of a flute note, resulting in a hybrid sound. However, this resulting hybrid sound might bear almost no similarity to the input sound. In essence, this is a form of "using neural networks the wrong way" or neural network bending (derivate of circuit bending). This is the basis of the software library, and the following sections will explain the various components of the library.

The table below shows the various sections of the library, this section will explain each section in detail. The sections are as follows:

Table 1: Music Representing Corpus Virtual (MRCV)

Section	Description
Neural Network 1	Music Generation
Neural Network 2	Sampler Instrument Procedural Generation (Sound Design)
Neural Network 3	Realtime Audio-to-Audio Inferencing (VST/AU plugin)
Neural Network 4	Neural Wavetable Generation through Mel-frequency Cepstrum Coefficients
Genere	Procedural Score Generation
Audio Dataset 1	Saxophone Ordinario Dataset
Audio Dataset 2	Saxophone Multiphonic Dataset
Audio Dataset 3	Piano Dataset
MIDI Dataset	MAESTRO Dataset

2.3 Neural Network 1: Music Generation

2.3.1 Introduction

The first neural network offered is a Multiple-In-Multiple-Out (MIMO) Neural Network model [Ayomoh and Ajala, 2012] or MixMo Neural Network model [Ramé et al., 2021]. Since the input and output of this neural network is MIDI, the neural network will have to be able to ingest the correct format for each column of data in the MIDI dataset. The chosen format of data is a list containing:

Onset Time Initial time value when note is played

Duration Time value when note is released (offset time - onset time)

Pitch MIDI pitch value

Velocity MIDI velocity value corresponding to loudness (and sometimes timbre)

This neural network only makes use of this four input variables. The given task, and by extension the loss function of the network, is to predict the next note in the series of notes. Functionally, we can express that as such:

$$\mathcal{L} = f(Note_{t+1} - Note_t) \quad \mathcal{L}_{components} \begin{cases} Onset\ Time \\ Duration \\ Pitch \\ Velocity \end{cases} \tag{5}$$

Since there are 4 separate components within the loss function, it will be hard to linearly combine these various components of loss to derive a single loss. Instead, the neural network can be made to predict 4 separate values. This is done by having 4 separate output layers, each with their own network and loss function. The loss function for each

output layer is the mean squared error (MSE) loss function. The MSE loss function is defined as:

$$\mathcal{L}_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{6}$$

2.3.2 Model

Conceivably, this leaves us with the motivation to construct a network that makes use of multiple inputs and produces multiple outputs. This is further informed by the fact that compositionally speaking. The various components of a note are not independent of each other. There might be interactions between each component of the note, for example, the onset time of a note might be dependent on the pitch of the previous note. With this understanding, a neural network can be constructed as such:

$$\begin{aligned}
sub\mathbb{M}_{pitch,duration} &= \left[\sum_{0}^{(L,\mathbf{p})}\right](x_{pitch} \cup x_{duration}) \qquad sub\mathbb{M}_{pitch,onset} = \left[\sum_{0}^{(L,\mathbf{p})}\right](x_{pitch} \cup x_{onset}) \\
sub\mathbb{M}_{pitch,velocity} &= \left[\sum_{0}^{(L,\mathbf{p})}\right](x_{pitch} \cup x_{velocity}) \quad sub\mathbb{M}_{duration,velocity} = \left[\sum_{0}^{(L,\mathbf{p})}\right](x_{duration} \cup x_{velocity}) \\
sub\mathbb{M}_{onset,velocity} &= \left[\sum_{0}^{(L,\mathbf{p})}\right](x_{onset} \cup x_{velocity}) \qquad sub\mathbb{M}_{duration,onset} = \left[\sum_{0}^{(L,\mathbf{p})}\right](x_{duration} \cup x_{onset}) \\
\forall \hat{y} = \{\hat{y}_{pitch}, \hat{y}_{onset}, \hat{y}_{duration}, \hat{y}_{velocity}\} &= \left[\sum_{0}^{(L,\mathbf{p})}\right]_i \bigcup_{i=1}^{n_{sub\mathbb{M}}} \binom{n_{sub\mathbb{M}}}{3_{\forall x}} sub\mathbb{M}_i
\end{aligned} \tag{7}$$

For further clarification, the notation used in Equation 7 is described in the figure below:

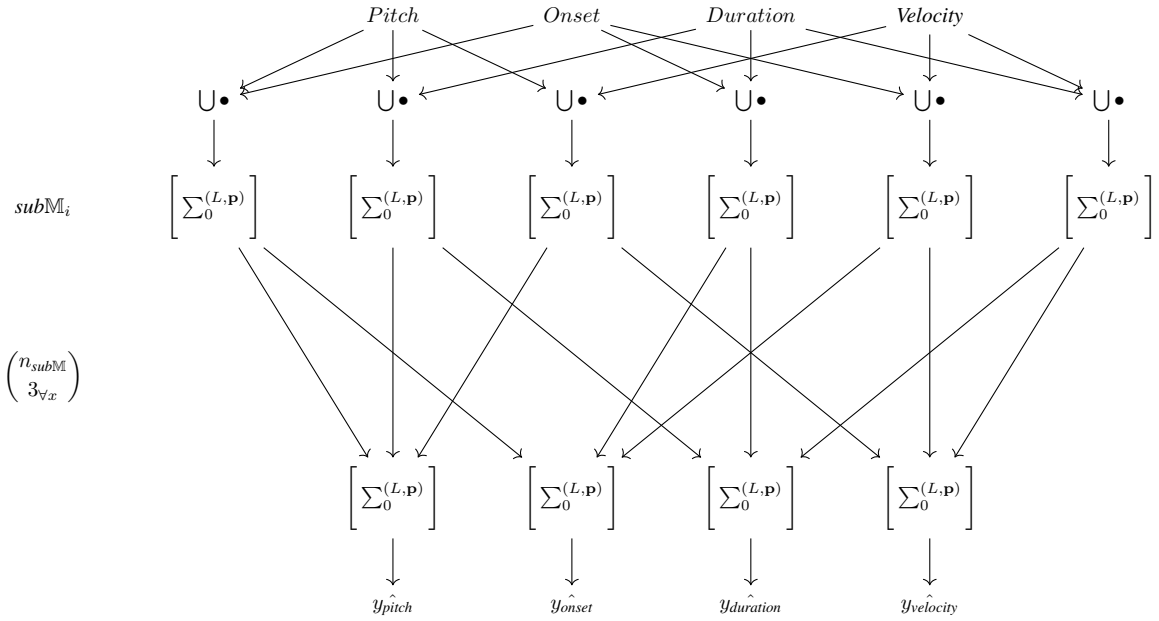

Figure 1: The structure of the Neural Network in Equation 7. With each input parameter connected to different submodels within the neural network. These submodels are then combined differently to produce the output parameters.

Clarifying the notation used in Equation 7: The $\cup$ operator describes any differentiable operator, which is inclusive of but not limited to the cardinal operators of $+, -, \times, \div$. Specifically, the operator used in the deployed version of the

system is the *concat* operator:

$$\begin{aligned} a &= [1, 2, 3] \\ b &= [4, 5, 6] \\ \textit{concat}(a, b) &= [1, 2, 3, 4, 5, 6] \end{aligned} \tag{8}$$

2.3.3 Dataset

This model is deployed with the MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization) dataset [Hawthorne et al., 2019], which is a collection of about 200 hours of virtuosic piano performances captured with fine alignment (3 ms) between note labels and audio waveforms. The recorded performances are MIDI recordings of virtuoso pianists performing of Yamaha Disklaviers. The repertoire is mostly classical (common practice period), including mostly composers from the 17th to early 20th century.

In the code, a lookup function has been made to construct the dataset. The lookup function takes in an array of strings and parses these strings to return a MIDI dataset with the corresponding composers. An example of this is as such.

```
midi_data = get_data_for_composer(main_data, ["Handel", "Medtner"])
# returns a dataset that has only the MIDI files of Handel and Medtner
# in each column the dataset contains:
startTimes = midi_data[:, 0]
endTimes = midi_data[:, 1]
pitches = midi_data[:, 2]
velocities = midi_data[:, 3]
durations = midi_data[:, 4]
```

2.3.4 Output of the Model

Here are some examples of the output of the model, represented as images of piano roll. The x-axis represents time and the y-axis represent pitch height (in midi note numbers).

(example image here)

2.4 Neural Network 2: Sound Design

2.4.1 Introduction

Continuing with the theme of misusing neural networks, Neural Network 2 is designed to be a sound design tool. The model takes in a sample of an audio at time $t-1$ and is asked to predict the sample at time t. An aspect that will be elaborated on further is the potential of mixing datasets. The model is deployed with the Saxophone Multiphonics Dataset as its default training data.

2.4.2 Model

The model is a simple dense-layered neural network with a dropout layer after every layer. The dropout layer will be notated as d in the model. Formally, we can refer to the mixture of Datasets $\mathcal{D}$ as:

$$\mathcal{D} = \bigcup \left(\{D_1, D_2, \ldots, D_n\} \right) \tag{9}$$

since the model is exposed to a mixture of datasets, the latent representations learnt by the model adhere to these datasets. The model will try its best to generate outputs that adhere to the latent patterns of these datasets.

$$\hat{y} = \left[\sum_{0}^{(L,\mathbf{p})} (d) \right] (x_{\mathbb{B}\ldots\mathbb{B}*}) \tag{10}$$

The model has certain network architecture parameters that are exposed to the user. In this case, layer count L and layer width p are exposed to let the user design the model architecture. $\mathbb{B}\ldots\mathbb{B}*$ refers to the block size of the input and output. More specifically, if the block size were 4, then the input would be $x_{t-3}, x_{t-2}, x_{t-1}, x$, and the next input would be $x_{t+1}, x_{t+2}, x_{t+3}, x_{t+4}$. The output of the model should be trying to return predicted $x_{t+1}, x_{t+2}, x_{t+3}, x_{t+4}$,

and $x_{t+5}, x_{t+6}, x_{t+7}, x_{t+8}$ respectively. For further clarification, this is the model architecture in the case of $L = 2$ and $p = 4$, with 1 input and 1 output from the dataset:

$$\hat{y} = \left[\sum_{0}^{(2,4)} (d) \right] (x_{t-3}, x_{t-2}, x_{t-1}, x)$$
$$\hat{y} \approx x_{t+1}, x_{t+2}, x_{t+3}, x_{t+4} \tag{11}$$

Graphically, this can be represented as:

For further clarification, the notation used in subsubsection 7 is described in the figure below:

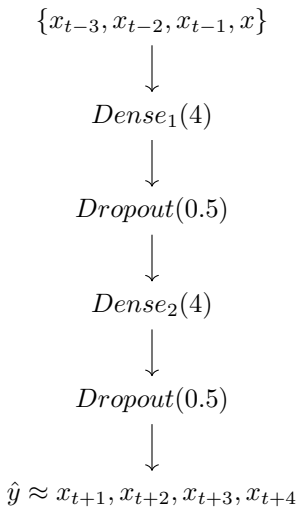

Figure 2: The structure of the Neural Network in subsubsection 7. With each input parameter connected to different submodels within the neural network. These submodels are then combined differently to produce the output parameters.

The model will then be asked to predict M number of blocks. Realistically, each block can be imagined as an audio file containing 44100 samples. Thus the final outputs of the model will resemble a batch of files of length (block size) with M number of files.

$$\{\hat{Y}\} = \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_M\} \tag{12}$$

These files are then passed to a script that will procedurally generate a Decent Sampler Instrument [Hilowitz, 2019], which is a sampler instrument that can be used in a Digital Audio Workstation (DAW). The script will procedurally generate a sampler instrument that will play the files in the order of the output. The sampler instrument will be generated with the following user accessible parameters: Attack, Decay, Sustain, and Release.

2.4.3 Dataset

The Saxophone Multiphonic Dataset (SMD) is a dataset of saxophone multiphonics. This dataset features a recording of all the multiphonics listed in the saxophone multiphonics book by Kientzy [1982]. SMD features PCM recordings at a sampling rate of 44.1 kHz. In total, there are 228 audio files.

2.4.4 Output of the Model

References

Elizabeth A Wilson. *Affect and artificial intelligence*. University of Washington Press, 2011.

Huimin Lu, Yujie Li, Min Chen, Hyoungseop Kim, and Seiichi Serikawa. Brain intelligence: go beyond artificial intelligence. *Mobile Networks and Applications*, 23:368–375, 2018.

John Howard. Artificial intelligence: Implications for the future of work. *American journal of industrial medicine*, 62 (11):917–926, 2019.

Maximos Kaliakatsos-Papakostas, Andreas Floros, and Michael N Vrahatis. Artificial intelligence methods for music generation: a review and future perspectives. *Nature-Inspired Computation and Swarm Intelligence*, pages 217–245, 2020.

Eduardo Reck Miranda. An artificial intelligence approach to sound design. *Computer Music Journal*, 19(2):59–75, 1995.

Koray Tahiroğlu, Miranda Kastemaa, Oskar Koli, et al. Al-terity: Non-rigid musical instrument with artificial intelligence applied to real-time audio synthesis. In *Proceedings of the international conference on new interfaces for musical expression*, pages 337–342, 2020.

Wallace Jackson and Wallace Jackson. The history of digital audio: Midi and synthesis. *Digital Audio Editing Fundamentals*, pages 11–17, 2015.

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

Eduardo Reck Miranda. *Readings in music and artificial intelligence*. Routledge, 2013.

Curtis Roads. Research in music and artificial intelligence. *ACM Computing Surveys (CSUR)*, 17(2):163–190, 1985.

MKO Ayomoh and MT Ajala. Neural network modeling of a tuned pid controller. *European Journal of Scientific Research*, 71(2):283–297, 2012.

Alexandre Ramé, Rémy Sun, and Matthieu Cord. Mixmo: Mixing multiple inputs for multiple outputs via deep subnetworks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 823–833, 2021.

Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=r1lYRjC9F7`.

David Hilowitz. Shop - decentsamples. `https://www.decentsamples.com/`, 2019. (Accessed on 05/02/2023).

Daniel Kientzy. Les sons multiples aux saxophones: pour saxophones sopranino, soprano, alto, ténor et baryton. *(No Title)*, 1982.