Parameter automation for granular synthesis

author:

Karol Bakunowski

supervisor:

Dr. Michael Zbyszyński

Submitted as part of the requirements

for the award of the Degree in

Music Computing

of

Goldsmiths University of London.

Department of Computing

Goldsmiths University of London

March 29, 2019

Abstract

My research is about stuff.

It begins with a study of some stuff, and then some other stuff and things.

There is a 300-word limit on your abstract.

A good abstract explains in one line why the paper is important. It then goes on to give a summary of your major results, preferably couched in numbers with error limits. The final sentences explain the major implications of your work. A good abstract is concise, readable, and quantitative. Length should be 1-2 paragraphs, approx. 400 words. Absrtracts generally do not have citations. Information in title should not be repeated. Be explicit. Use numbers where appropriate. Answers to these questions should be found in the abstract: What did you do? Why did you do it? What question were you trying to answer? How did you do it? State methods. What did you learn? State major results. Why does it matter? Point out at least one significant implication.

Acknowledgements

Acknowledge all the things!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Aims and Objectives

The initial aim of the project was to implement a machine learning solution to the task of granular synthesizer programming based on sound matching. Consequently building a tool that would assist musicians in creating interesting sounds, provided an audio input to the system.

In the process of building this tool, the project extended into more technical direction, with a focus of creating a usable tool, despite some shortcomings in the areas of machine learning. More specifically, a large amount of time was spent configuring the three main pieces of the project: the synthesizer, audio analysis and machine learning to work together as smoothly as possible, rather than making each of them exceptional on it's own.

The methods here differ from other approaches in literature (cite some papers that have done this, eg. Matthew, Leon) primarily in that here, one standalone tool has been created, that is usable on it's own, directly by the user. A usable and focus directly on the interaction between user and the machine. Direct access to predictions and straightforward usability, handled by two buttons.

The main objectives are:

1. To build a tool that is helpful to artists in creating new sounds

2. To challenge the interaction between an artist and a preset as a starting point to synthesis

3. To achieve a response that is not only stimulating to the user but also differs from simply randomizing the parameter values

In measuring the project's success, the subjective sonic coherence and similarity of predicted outputs may be considered the best indicator (the human discriminator?). Along with that, a more quantitative analysis, such as comparing the audio descriptors on target and predicted sounds should prove itself useful, as it will allow to do a quite generalisable, and objective comparison of target vs predicted sounds.

In (future chapters) I provide some critique on established techniques of assessment of these types of problems and ultimately conclude that some stuff in the best way of doing it.

1.1.1 Deliverables

To achieve that, I propose a standalone application, that combines all three main aspects, and lets the user set parameters of a synthesizer based purely on the microphone input to the system. Synthesis, implemented with the 'JUCE' framework. Audio analysis tools, implemented with the help of the 'Essentia' library, and an implementation of a Multilayered Perceptron feedforward neural network, done in 'Keras'. Together with the help of the 'Frugally Deep' library, responsible for deploying the Keras model into C++ code, the three modules were able to work together inside of the synthesis program.

Chapter 2

Literature Review

2.1 Problem background

Mimicking sounds is very intuitive for humans. Humming along a song we've heard previously, or generally repeating something so that it sounds as close to the original as possible seems like a rather trivial task for us. However, musicians who are able to translate what they hear in their head, directly to an instrument, be it a physical one like piano or guitar, or a software based instrument, are considered viruosic.

Yet, many successful musicians might be programming synthesizers following a not clearly defined 'intuition', and often look for inspiration for the sound they are trying to create. Especially today, a lot of sounds are created by the means of experimental methods, often using presets as a starting point.

Therefore, to be able to replicate any sound in a synthesizer, based only on an audio input would seem like a very intuitive way of working, and interacting with synthesizers.

2.1.1 Audio descriptors

Unfortunately, based on how sound is represented digitally, mimicking sounds is not as intuitive for computers as it is for humans. In order to describe sounds we need some algorithms, that are capable of generating meanigful data about given audio signal. A great amount of these analysis tools exist, and generally they can be devided into two categories. One dealing with timbre, and another dealing with rhythm, or otherwise temoral qualities of the signal.

2.1.1.1 Rhythm

Onset detection, which tries to estimate how many ‘peaks’ there are in a signal is a very useful algorithm to estimate how much rhythmical content there is present.

2.1.1.2 Timbre

In order to compute any sort of pitch detection, the audio signal has to be translated from the time domain to the frequency domain. This can be achieved by the inexplicably useful algorithm in music today: Fourier Transform. It can dissect a signal to it’s most basic sinusoidal components, therefore determining how much of which frequencies are present in the signal, which in turn can directly translate into pitch.

Building on that, possibly the most powerful algorithm for determining the timbre of an audio signal is Mel-frequency cepstrum coefficients, or MFCC. Many studies have been done to prove the usefullness of MFCCs in describing the timbral, and as a matter of fact rythymical content of audio signals. They are mainly used in speech symthesis and speech recognition, however can be apply to any signal, and are a very powerful descriptor.

2.1.2 Granulation

There is one, major limitation, when it comes to parameter predictions for a granular, or any other corpus-based synthesizer. The sounds created are heavily based on the sample fed into the synthesizer. Each parameter changes meaning significantly, once the input sample is changed.

Two possible solutions come to mind, when trying to overcome this problem. Using one input sample for synthesis, and trying to predict samples for one perticular instance of the granulator is one. Another, would be trying to fit some universal analysis algorithms, that could decribe rhythym, pitch, density of sound etc. independently of the original sample. This way any sample could be ‘molded’ into what would resemble the original prediction target.

Concatenative synthesis is an alternative approach to this problem. It could be beneficial, as it would try to find ‘grains’ as closely resembling parts of the target as possible, and recreate it using little parts, almost like puzzles, that the algorithm thinks fit. However, that approach also has limitations, as it could only recreate the target sound out of the samples stored in it’s database, therefore making the output biased.

2.1.3 Machine Learning (predictions?)

A neural network could be used to build a parameter space for any synthesizer from some input, which here would mean the audio analysis algorithms results. A lot of research has been done on this topic. From feedforward networks (cite) paired with analysis algorithms, to Long Short Term Memory networks to take into account the temporal information conveyed in different frames of the MFCC analysis (citation). Convolutional networks are also very often used, especially for classification of sounds.

Different algorithms were used previously to tackle the task of automatic synthesizer programming, including Hill Climber (cite), Genetic Algorithms (cite) etc.

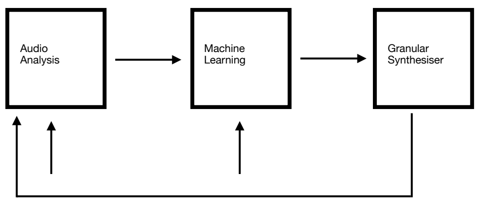
Alternatively, in order to try and generalise the predictions for different input samples for the synthesizer. The predictions could be devided between different parameters. Then linking different prediction algorithms with different parameters could possibly allow for a creation of a more direct relationship between audio descriptors and the synthesis results.

Chapter 3

Methods

The method of tackling this problem will be based on previous research done in this domain. There are three essential components for this project:

Figure 3.1: Basic flowchart



The box 'Granular Synthesiser' in figure 1. represents a granular synthesizer. It was implemented in C++, using the 'JUCE' library.

'Machine Learning' stands for the predictions module, or more precisely the Multilayer Perceptron model built in 'Keras', and later ported to C++ with the help of the 'Frugally Deep' library.

Lastly, the 'Audio Analysis' square represents the aspect of the project responsible for extracting audio features from input sounds. This was done with the 'Essentia' library, that allowed me to perform frame cutting, windowing, extracting the spectrum and finally MFCCs on any specified buffer.

3.1 Audio Analysis

The only algorithm used for describing audio is the Mel-frequency Cepstrum Coefficients. As mentioned in the Literature Review section, it is proven to be one of the most reliable desriptors available, and can not only describe the temporal qualities, but with the right treatment, also be indicative of the changes that happen in the input signal. More on that later.

In order to extract MFCCs from the signal, some preprocessing has to be done. Namely, the signal has to be split into frames, on which the FFT algorithm is performed. Each of the frames has to include a little from the previous and the future one, in order to give a better representaion of the signal. This is called a hop size.

Then, for each frame, MFCC are calculated based on the extracted FFTs. The result for one frame is a vector of 13 floating point values, each corresponding to a different range of frequencies.

At a sampling rate of 44.1kHz, and a buffer size of 512 samples, this corresponds to exactly 45 vectors of 13 float values for one second of sound.

3.2 Predictions

3.2.1 Dataset

In order to get any predictions, a dataset, linking the MFCC values with the synthesis parameters had to be created. Initially, I have set out to create a data set of every possible combination of parameter values, and their corresponding MFCCs. However, even though there are only 5 adjustable parameters, with a static hop size of a tenth of each parameter, it would take more than 1000 hours to complete this task, given that each parameter combination would be sampled for exactly one second.

In order to evade this limitation, a different approach was taken. Every second, each parameter was set to a random value between their unique minimum and maximum values. Consequently, the audio features were extracted on each of these iterations, for exactly 10000 instances, which allowed for a creation of a dataset, of 10000 different parameter values, each corresponding to 10000 2D vectors, of size 45 x 13 each.

3.2.2 Feedforward neural network

The neural network implemented for the task of predicting the synthesizer parameters is a Multilayer Perceptron. It's a relatively simple, feedforward network, consiting of 3 layers. The 45x13 MFCC vectors served as the features, and a vector of 5 different parameter values as the values to be predicted by the network. Firstly, the 2D vectors, were flattened, like an image would be in a convolutional network.

Trained for 2000 epochs... accuracy of 50%... parameters = this and that... pretty shit, and not working very well, put not enough time for more experimentation...

3.3 Synthesis

3.3.1 Granular Synthesis (basic explenation of the granular synth concept)

Describing granular synthesis in one paragraph is impossible without cutting some edges, and leaving out many intricacies of the technique. However, a summary of sorts, describing the main concepts behind the algorithm will be attempted here.

Granular synthesis is an algorithm... Grain clouds... Xenakis... Rodes... Some other folks... Capable of many thing, versatile... graphs, graphs, graphs and pictures...

3.3.2 The JUCE implementation (author's implementation)

The granular synthesis algorithm was implemented using the 'JUCE' library. It is one of the most established libraries in the professional audio community, being used by companies such as 'Cycling 74', and 'Korg' amog others(cite website?). On top of all things audio, it provides a convenient way of creating a graphical user interface, which was an important part of this project. Thanks to 'JUCE', the implementation was kept minimal, and elegant.

explenation of implementation... grain class... how grains are called... how they are altered by the dials... show code snippets...

3.3.3 Synthesizer parameters

There are 6 adjustable parameters... position... ** where to spawn new grains grain size... ** how the long are the grains spawned position offset... ** how far away from the original position should the grains be spawned number of grains... ** how many grain should be active at one point in time pitch... ** how higher lower the pitch of each grain should be relative to the original sample global gain... ...

Chapter 4

Results and Analysis

Chapter 5

Discussion

5.1 Summary of Findings

5.2 Evaluation

5.3 Future Work

Chapter 6

Conclusion

Appendix A

An Appendix About Stuff

(stuff)

Appendix B

Another Appendix About Things

(things)

Appendix C

Colophon

This is a description of the tools you used to make your thesis. It helps people make future documents, reminds you, and looks good.

(example) This document was set in the Times Roman typeface using LaTeX and BibTeX, composed with a text editor.

Bibliography

[1] Anne Author. Example Journal Paper Title. *Journal of Classic Examples*, 1(1):e1001745+, January 1970.