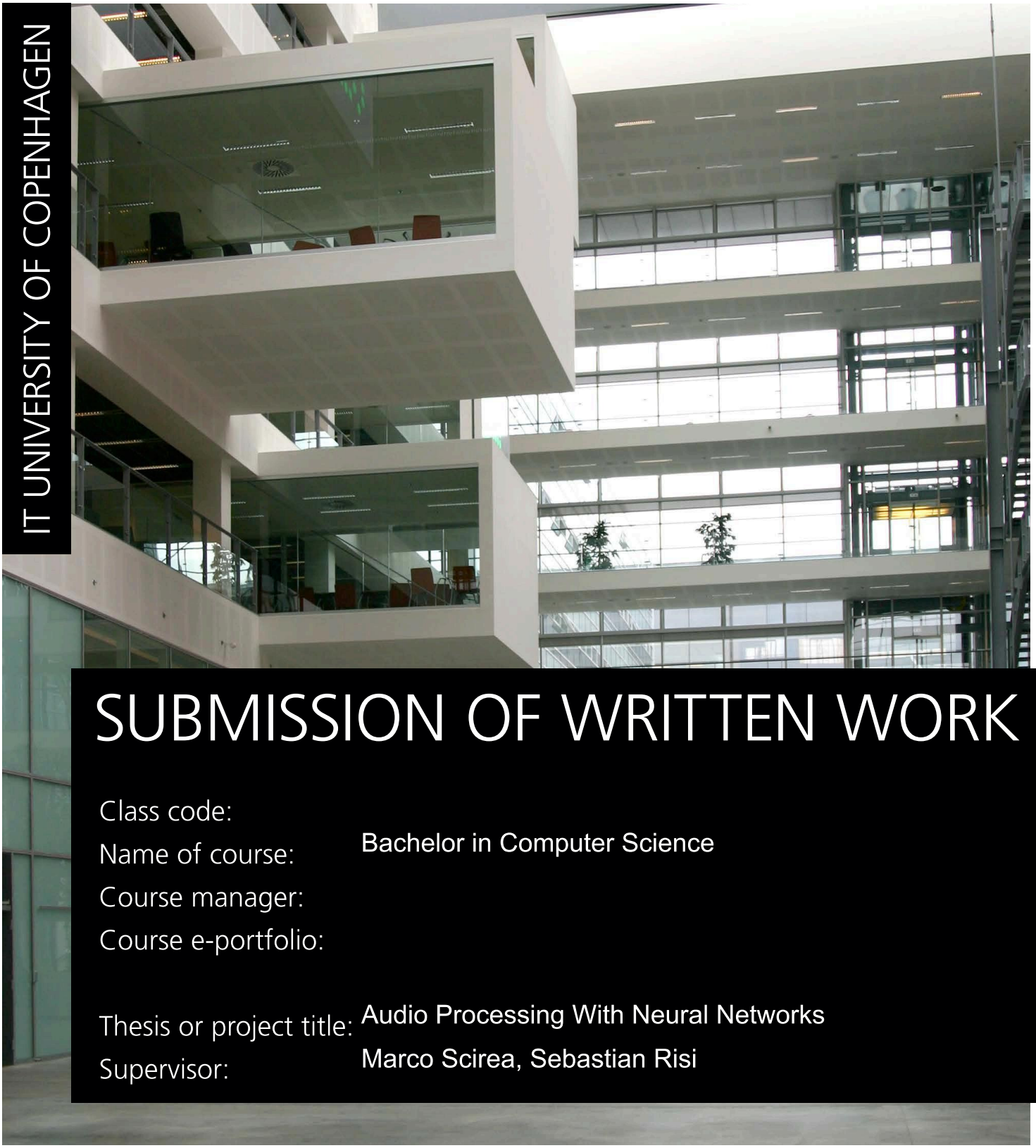
IT UNIVERSITY OF COPENHAGEN



SUBMISSION OF WRITTEN WORK

Class code:

Name of course: Bachelor in Computer Science

Course manager:

Course e-portfolio:

Thesis or project title: Audio Processing With Neural Networks

Supervisor: Marco Scirea, Sebastian Risi

	Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1.	Nikolai Storr-Hansen	17/02-1968	nsth@itu.dk
2.			@itu.dk
3.			@itu.dk
4.			@itu.dk
5.			@itu.dk
6.			@itu.dk
7.			@itu.dk

Audio Processing with Neural Networks

Real-time Audio Processing with Interactive Evolutionary Computation

Bachelor thesis

May 20, 2015
ITU, Copenhagen
Software Development Line

For graduating with a bachelor of Computer Science

By Nikolai Storr-Hansen

Supervisors:

Marco Scirea, Sebastian Risi

Acknowledgements

I would like to thank Björn Þór Jónsson for writing Breedesizer and for letting me use it. I would like to thank Marco Scirea and Sebastian Risi for their invaluable help and support in realizing this project.

Copenhagen, May 20, 2015 N.S.H

Abstract

Waveform shaping is used in such varied areas as signal processing, active sensing and entertainment. This project will explore the use of evolutionary computing as a means of producing audio effects on real-time audio input. In particular, how a Compositional Pattern-Producing Network facilitates a wider range of sound effects than achievable by conventional means. Using interactive evolution computation to shape the waveform, the user is provided an infinite number of effects to choose from. This offers a novel addition to the common audio effect unit, typically limited in its expressive range. This project modifies Breedesizer - a project using CPPN to produce the timbres of a synthesizer - into working with streaming audio to produce amplitude modulation with CPPN. A survey has produced positive feedback wrt. the quality and the added variety of the effects produced by the developed system. This use of evolutionary computing may be utilized in the production of audio effects and for audio effect unit production with e.g. the music and entertainment industries.

Contents

Introduction

STANDARD sound effects for analogue instruments in music production (e.g. guitar pedals) often use waveshaping [6] to achieve an audio effect - e.g. distortion, compression, chorus etc. Typically, user-adjustable values for volume, effect amount and perhaps a mix parameter to adjust the amount of 'clean' audio in the final output is supplied. This basically provides for a single variety of audio effect. Though it is possible to combine effects, as a beginner it is still a complicated and expensive process to achieve a satisfactory result. The ability of neural networks [16] to produce infinitesimal variations on a theme leads to the idea of using them in sound processing. An audio effect unit based on neural networks could provide an unlimited range of effects for the user to choose from. This would produce a highly customizable effect unit with endless possibilities for personalization.

This project will use evolutionary computation to explore the possibilities for developing audio effect units on audio input. The system developed will generate a number of CPPNs, corresponding to sound effects on the input audio. The user chooses the best individuals from which the system will generate the next generation of audio effects. The result is reached when the user is satisfied with the resulting sound. This project pose the following questions: can neuro-evolution make creating audio effects (guitar effects) easier? Can neuro-evolution provide a higher range of effects than otherwise possible? Can we use CPPNs to evolve new significantly different sound effects?

Artificial neural networks (ANN) [2] are algorithms inspired by biological neural networks often used in machine learning and pattern recognition. They consist of a system of interconnected 'neurons' connected by weights, the weights being equivalent to human synapses. The weights are 'learned' through experience in an iterative procedure [16]. One way for a network to learn is through reinforcement: the network takes a reward/penalty signal and the weights are then changed to achieve an I/O behaviour which maximises the probability of reward and minimises the probability of a penalty.

Using evolutionary computation for producing art is not an entirely new concept. Notable examples include PicBreeder [14], [26], and Soundbreeder [34]. Other interesting projects exist, [4], [10] . This project is based on Breedesizer [22], a synthesizer [27] that creates waveshapes by using NeuroEvolution of Augmenting Topologies (NEAT) [28], and Compositional Pattern-Producing Network (CPPN) [29] to create frequency modulation (FM) synthesis [1].

In Breedesizer and other art projects involving neural networks the user is the objective function of the neural network. Each sound is evaluated in a process of Interactive evolutionary computation (IEC) [30], in which a user selects the most pleasing sounds, as parents for the next generation. i.e., the user functions as the fitness evaluator effectively 'flipping' through generation after generation deciding which ones shall offspring the next generation, until satisfied with the result from a subjective point of view.

1 Background

This section will look at previous research into the application of evolutionary computation in audio production. It will function as a discussion on its potential for producing sound effects suitable for audio production.

1.1 Evolutionary Computation in arts

Evolutionary computation in arts is not new [1], [2]. A quick look at the internet reveals a plenitude of experiments into image, music and 3d generative content. Books and theses has been written on the subject [25] , [11], [32], [13], [12]. Evolutionary computation in music has traditionally been utilized in connection with synthesizers, [18], [12], [13], and algorithmic composition [33], [20]. Evolutionary techniques have also been utilized in graphics [14], [26], and audio [34]. Mainstream artists such as Portishead [3] and Radiohead have used these technologies in their music production.

One area where evolutionary computation is not over-represented is in connection with analogue instruments and live audio. There might exist a potential in this area for developing applications for the music industry. This project will apply evolutionary computation on real-time audio. As such, this project is more aimed at the existing music industry than the social media music production currently on the rise. 2.4.

1.2 Evolutionary computation and Audio

Evolutionary computation in relation to audio can be explored in many directions. It can be used to compose music, it can be used to classify audio, e.g. a phrase, a score or a genre. It can be used to evolve parameters in some processing unit. The list is long and open-ended.

[1] http://en.wikipedia.org/wiki/Evolutionary_art
[2] http://www.red3d.com/cwr/evolve.html
[3] https://www.facebook.com/portishead

A plenitude of work exists on the subject [10], [31], [15], [35], [21], [8], [4]. One of the more well known examples of evolutionary computation in music is the project Soundbreeder [34].

This project is based on Breedesizer [22]. Breedesizer uses a neural network to shape simple waveforms (sine waves) which are then used to modulate the frequency of the oscillators of a synthesizer. Breedesizer effectively uses neural networks to create the timbres of the waveforms of a simple synthesizer.

2 Analysis

This section investigates the parts used in the project. It begins with a short overview of the project, then a description of the Breedesizer project is given followed by a brief description of neural networks, then the Web Audio API is described followed by a section on wave-theory.

2.1 Overview

The project takes as input a guitar or other instrument (e.g. a microphone). The input audio, thus, comes from the computer microphone or, if available, a USB input port. The input audio is modified in two ways: first, the amplitude of the input signal is modulated by the CPPN networks and, second, the Web Audio API 2.4.1 is used to provide additional ordinary audio effects like compression, distortion and reverb. The final audio is sent to the computer's sound-card. A user interface provides interaction with parameters of the CPPN evolutionary computation effect and the standard effects. The final result can be recorded and the CPPN parameters output in JSON format.

The project is based on the Breedesizer project which is modified to work with audio instead of midi. the CPPN networks produced by the Breedesizer project are reused without modification. In particular, the wavetables which are the output of the CPPN networks are used here to do audio modulation.

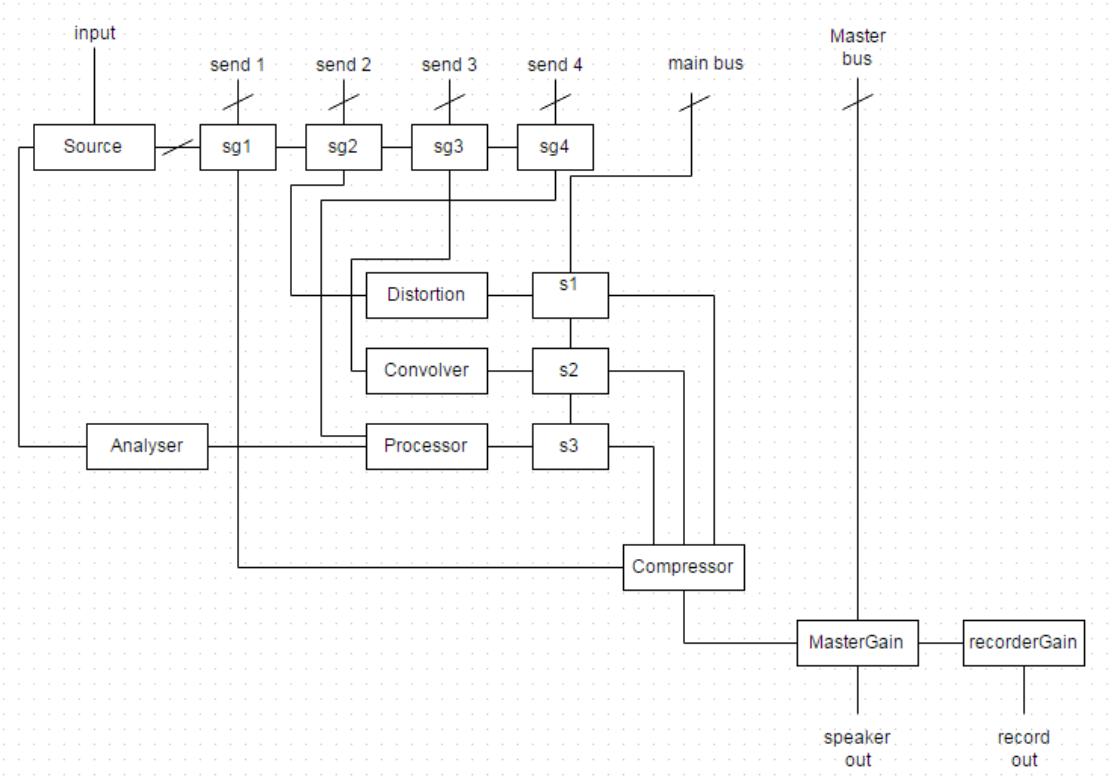


Figure 2.1: Project audio routing graph

Figure 2.1 shows the project's routing scenario (see section 2.4.1). It consists of 4 sends and 3 submixes. Each send allows one effect unit. Each submix controls the amount of the effect. This achieves explicit control over the volume, or "gain", of each individual effect in the overall mix of signals. Input audio is stored in buffers (the 'source' node in the figure). The processor node then allows manipulation of samples from the buffer.

2.2 The Breedesizer Project

"The aim with Breedesizer is to explore the feasibility of utilizing the powerful properties of Compositional pattern-producing networks evolved through NeuroEvolution of Augmenting Topologies (CPPN-NEAT), for encoding structure, in the formation of unique waveforms to use in the construction of oscillators, producing interesting timbres" [22]

The Breedesizer project explores the CPPN-NEAT neuro-evolution technique [29], to produce waveforms used to evolve timbres [5], [7]. Periodic waves are used to construct wavetable oscillators [3] producing timbres that is evaluated in a process of Interactive evolutionary computation (IEC) in which humans select oscillators as parents for the next generation of wavetable oscillators [30].

Basically, CPPN-NEAT is used to produce populations of phenotype waveforms, where each

individual is backed by a genotype composed of one evolved CPPN network.

This technique is used to build a synthesizer [17]. In connection with each wavetable oscillator a simple Frequency Modulation (FM) [23] synthesizer is constructed. This is done by forming two periodic waveforms with outputs from the evolved neural networks. Those are then used to construct a wavetable oscillator, one acting as a modulator of the timbre produced by the other, acting as the carrier. Breedesizer is constructed in Javascript with the Web Audi API. The project uses the dsp.js [9] signal processing library to make Fourier Transforms in the creation of wavetables [19]. The Fourier transform is a well known algorithm used in as varied fields as quantum mechanics and spectral analysis of time-series. In audio production Fourier transform is used for transposing between a waveform's frequency domain and its time domain.

This project initially attempted to follow the Breedesizer example and make frequency modulation on an audio input. This attempt was cut short in favour of the programmatically easier but more processor-intensive amplitude modulation.

2.3 Evolutionary Computation

Neuroevolution is an approach to artificial intelligence motivated by the evolution of biological nervous systems. By applying evolutionary algorithms neuroevolution can construct artificial neural networks. i.e., the abstractions of natural evolution is used to abstract biological neural networks. Neuroevolution can be seen as a means of investigating how intelligence evolves and as a method for engineering artificial neural networks used to solve some task.

Common artificial neural network learning algorithms evolve through supervised learning. They therefore depend upon existing training data. In many areas, such as games and robotics, optimal sequences of actions are not always known. Only after the fact is it possible to observe how well a succession of actions worked. Neuroevolution can produce a network that optimizes behaviour given sparse feedback and no direct information about what to do.

Common applications of neuroevolution are: reinforcement learning, evolutionary robotics, and artificial life. Applications include evolving behaviours for board games and video games, controlling mobile robots, and investigating the evolution of biologically-relevant behaviours.

Artificial neural networks (ANN) consists of a system of interconnected 'neurons' connected by weights (connection strength). The weights can be seen as the equivalent to biological synapses. The weights are trained through experience, typically in an iterative procedure [16]. One way for a network to train is through reinforcement: the network takes a reward/penalty signal and the weights are changed to achieve an I/O behaviour which maximises the probability of reward and minimises the probability of a penalty.

Many neural learning methods modifies only the connection weights between nodes. Neuroevolution allows optimization of additional parameters, such as the network structure (e.g.

adding neurons or connections), computation type of the individual neurons, and learning rules.

Basic networks usually consists of input and output layers with a number of layers in between them. These inner or 'hidden' layers is where the computation takes place. A feedforward network consists of one-directional layers of interconnected neurons. The input to the network is carried forward from one layer to another, each layer performing computation before handing information on to the next layer. By changing the weight of the connections between nodes the net can learn to map from input vectors x to output vectors y.

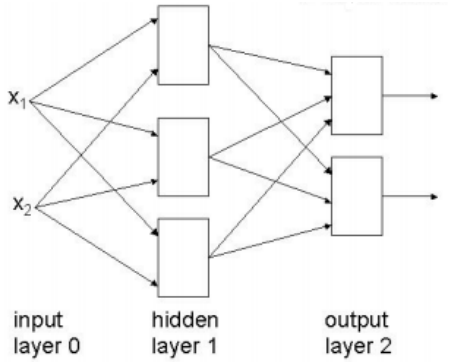


Figure 2.2: A two-layer feed forward network.

A trained network can generalize. e.g. a net can be trained to associate certain letters with target vectors indicating which one. The same network connects differently to express different letters. For instance, the network recognises the letter T so the output of the neurons of T (T-class) are greater than those of C (C-class).

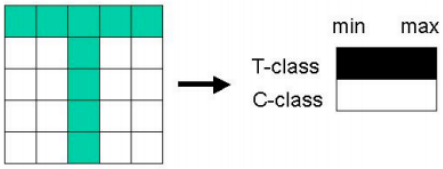


Figure 2.3: a trained network associating an input with the letter T.

The net recognises that the pattern below is more T-like than C-like. Consequently, the neurons of T-class output should be greater than that of the C-class. It is in the sense that a network can generalize that we say it is able to learn.

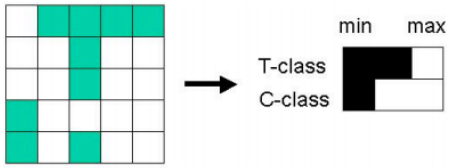


Figure 2.4: a trained network choosing between the letters T and C

Node connections are weighted differently. When a node receives a signal from several nodes at once the weights of their connections determine the response, usually by using some transfer function. e.g. taking the sum or an average over their weights in combination with the receiving node's previous value. The node decides on a value which it then uses to trigger the next set of nodes with. The output nodes decides what to do with the final output.

2.3.1 Back Propagation

The neural network can learn its own connection weights. This is done by a process called back-propagation. The network start out with a set of random connection weights. Next, for a given set of inputs (node values) we give the network a desired set of outputs. The network then calculate an output based on its initial random connection weights, which is likely nowhere near the desired output. The difference between the desired output and the calculated output is called the network error. The output nodes and the nodes they are connected to use the network error to adjust the connection weights between them. The new weights are calculated by an equation based on the old weight, the nodes input value, the error and a learning-rate (a parameter that changes the bias between learning accuracy and learning speed, a momentum). These nodes then use their newly calculated network error to do the same thing with the nodes they are connected to, and so forth until eventually the entire network has had its connection weights adjusted and an error assigned. The idea is to decide which nodes are most to blame for the error in the output and to try to adjust their weights the most.

In effect, the connection weight adjustment is pushed back through the network from the output to the input until all weights are adjusted. Then, the network tries the original input again. The calculated output should be closer to the desired output. This process of trial and error continues until the desired output is achieved. This is a slow process. To speed up the process we can adjust the learning rate or change the amount of nodes.

2.3.2 Activation functions

In neural networks the activation function of a node defines the output of the node given an input or inputs. A step function is one of the most basic activation functions. In its most basic capability the output of a step function is a certain value A_1 if the input sum is above a given threshold and A_0 if the sum is below. A linear combination is another kind of activation function in which a weighted sum determines the output.

A common activation function for neural networks including NEAT, is the sigmoid function. The sigmoid function is similar to the step function but with the addition of a region of uncertainty. The derivative of sigmoid functions are easy to calculate making them useful in calculating weight updates in some training algorithms. Many other activation functions are possible. e.g. Gaussian or periodic functions.

Unlike other neural networks CPPN includes a field for specifying the activation function. When a node is created it is assigned an activation function from a set of functions. The choice of function can bias the network toward specific patterns. The pattern producing characteristic of CPPNs are easily seen in graphical applications [26].

Breedesizer and this project uses four activation functions, each with equal probability for being picked. It might be possible to achieve an interesting result by biasing the choice of activation function towards a particular type. i.e., one pattern might produce better sounding effects than another. This idea is left as a future endeavour.

2.3.3 NEAT and CPPN

Compositional pattern-producing networks (CPPNs) is a method developed by Kenneth Stanley as a variation of ANNs that, as stated, use a variety of activation functions at each node [29], [24]. The choice of functions can bias toward specific types of patterns and regularities.

> *"... periodic functions such as sine produce segmented patterns with repetitions, while symmetric functions such as Gaussian produce symmetric patterns. Linear functions can be employed to produce linear or fractal-like patterns."* [1]

The basis for CPPN is the NeuroEvolution of Augmenting Topologies (NEAT) algorithm. NEAT is a genetic algorithm for generating evolving artificial neural networks, developed by Kenneth Stanley in 2002 [28]. NEAT begins with a population of small, simple networks, that are complexioned over generations [28]. Unlike a feedforward network NEAT connections may go backwards. Another difference is that NEAT has no hidden layers between the input and output layers. Neurons between the input and output layer evolves freely. NEAT alters the weighting parameters as well as the structures of the network to find a balance between the fitness of the evolved solutions and their diversity. In effect evolving both connection weights and the topology of the network (nodes and connections).

NEAT does not use back-propagation to learn the neural network. Instead it uses two genetic operators, crossover and mutation. Crossover is when two (or more) parents mate to produce offspring. Mutation is asexual reproduction where the offspring is a mutated clone of a single parent. A NEAT mutation can affect either the network structure or its weights. In NEAT, mutating the structure of the network can mean adding or removing a node or a connection.

Compositional Pattern-Producing Networks mimics natural development to achieve the same representational efficiency in computers as when DNA represents complexity in living organisms. CPPNs are based on developmental encodings, i.e. encodings that maps genotypes to phenotypes through a process of continuous growth from a small starting point to a mature form. By facilitating the reuse of genes a small set of genes can encode a much larger set of

[1] http://en.wikipedia.org/wiki/Compositional_pattern-producing_network

structural components. CPPNs take advantage of the particular properties of NEAT method which enables evolving increasingly complex neural networks over generations. By basing CPPNs on NEAT it is able to efficiently evolve increasingly complex phenotype patterns based on a smaller genotype.

2.3.4 Why CPPN?

One reason to use CPPNs in this project is the variety of audio effects CPPN makes possible. In contrast to the traditional effect unit, a CPPN-based system can provide in endless amount of different effects, each tailored to the users wishes. By utilizing interactive evolution (in which the user determines which individuals will be the parents of the next generation) the experimenter can intentionally explore the generated patterns.

The experimenter is presented with the patterns of the generations and can then pick the parents she sees fit for the next generation of effects. Selection is entirely determined by a human experimenter. This makes interactive evolution a suitable method for exploring the characteristics of various encodings. An experimenter can explore implicit capabilities such as elaboration and preservation of regularity. Desired properties like symmetry will be immediately apparent to a human experimenter, who can pursue it explicitly. The interface of the developed system therefore allows the user to adjust the probability of adding nodes and node connections in addition to adjusting the amount of mutation on each individual (see section 4.1.1) thus exploring the properties of elaboration and preservation.

In the case of this project the desired properties are audial but the same principle apply in e.g. graphics. An experimenter will immediately recognize the desired properties and be able to quickly pursue a certain goal.

2.4 Audio production

Traditionally, professional music applications have been marketed at the professional music production community and mainly running on dedicated workstations. The past decades have seen an increase in the development of music production applications for all sorts of devices, partly fuelled by technological development, market development and industry competition [2], [3]. Smart-phones and tablets has allowed a new market for music production software highlighting casual and social qualities of the applications. Some applications for tablets and smart-phones are even used in professional music production, such as remote controlled 'front-of-house' applications [4], [5], [6] which hooks up wirelessly to a mixerboard, allowing the

[2] http://www.musictech.net/1904/04/ios-music-making-apps-the-ten-best-fl-studio-hd/

[3] http://lerablog.org/technology/software/best-and-must-have-music-making-apps-for-smartphones-and-tablets/

[4] http://www.infocomm.org/cps/rde/xchg/infocomm/hs.xsl/36282.htm

[5] http://www.behringer.com/EN/Products/X18.aspx

[6] http://www.kicktraq.com/projects/132274466/wi-fi-audio-mixer-control-with-smartphones-and-tab/

soundman to walk around the venue while working the sound from a smartphone or tablet.

In line with this development, the introduction of the Web Audio API [7] in 2011 advanced web-based music systems by using JavaScript to enable real-time sound synthesis in web browsers. the goal of the API is to incorporate the capabilities found in modern music software such as sound synthesis, mixing, and processing tasks into the web browser without additional third-party components like Flash or QuickTime.

Web-based music production sits between professional and mobile applications free from regulation - i.e., App Store, etc. - and the user experience shows potential. In terms of real-time processing, there are some downsides as opposed to native execution, latency key amongst those.

Building atop the highlevel Web Audio API, many API's like WAAX [8] (Figure 2.5) expands on the notion of web-based audio production making the capabilities stretch well into previously proprietary domains.

Many existing applications built with these tools have in common that they cater for synthesizers. i.e., they use not processor-intensive oscillators to produce sound. This gives the appearance of zero-latency processing. This project attempts to push the limits of web-based audio processing by combining multiple effects, including CPPN which uses a separate API for Fourier transform, with real-time input and amplitude modulation on audio buffer samples.

[7] http://webaudio.github.io/web-audio-api/
[8] http://hoch.github.io/WAAX/

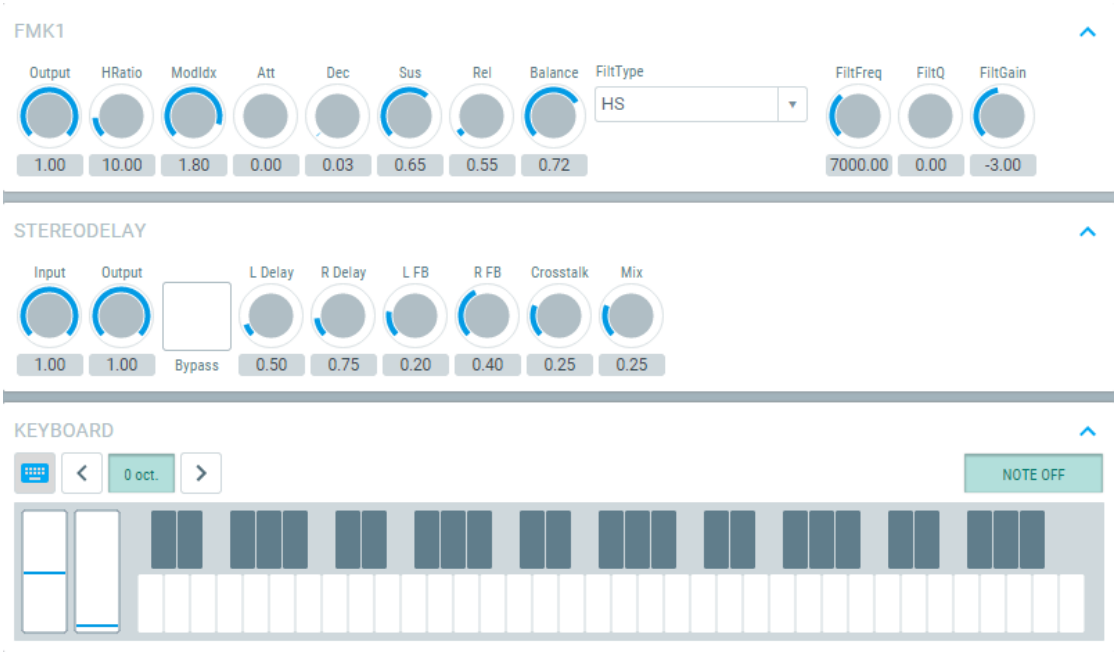


Figure 2.5: An application build with the WAAX API (build on the Web Audio API), origin: http://hoch.github.io/WAAX/).

2.4.1 Web Audio API

The Web Audio API is build on the abstraction of Nodes. A node forms a single logical processing unit. Nodes are linked together to form an audio routing graph via their inputs and outputs, forming a chain that starts with one or more sources, goes through one or more nodes, and ends up at a destination - e.g. the computer's sound card-.

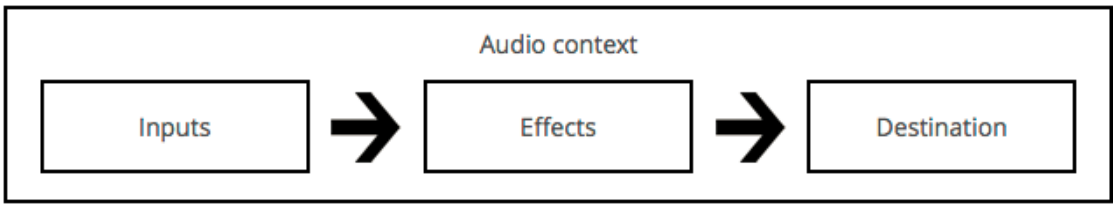


Figure 2.6: Node graph abstraction

This high-level API abstracts the physical music production industry standards. Software nodes are connected in a chain much like, in the real world, an instrument is connected to an amplifier which is connected to a mixer which sends the audio to the speakers or a recording unit. To provide control over the effects another technique from the music industry is utilized, sends. A send is a separate channel to which an effect is connected. The inputs are connected each to a channel from where the audio can be routed through sends to achieve effect on the signal. Varying the amount of a particular send varies the amount of that effect on all audio signals routed through the send. Varying the amount of that send on the individual channels

varies the amount of effect on that particular audio signal. This is a common strategy and exists in musical equipment like pedals and mixers, some powered speakers a.o. [9] [10].

One of the creations of the Web Audio API is its *scriptProcessor* node [11] which provides a stream of audio samples (PCM) from the microphone (or USB) input which can then be manipulated.

This project utilizes the *scriptProcessorNode* to manipulate audio samples. When audio is streaming the Web Audio API's *audioBufferSourceNode* is filled with audio samples. The project connects (arithmetically) input samples with equivalent CPPN samples to produce audio effect. As long as there is input the processing continues.

The web audio API has a 'fire and forget' philosophy. e.g. once all references to a node has been dropped it will be garbage-collected.

2.5 How to Measure the Result

This project develops a system for evolving audio. In this sense it exists in the realm of art. Art is not measurable in a scientific sense. So how do we measure the results of the project? We cannot talk about measuring quality or breadth or height, or any other scale of measure. When it comes to art the measure is subjective.

The choice of which offspring should parent the next generation of audio effects is a subjective one [30] but we can compare sound samples from different generations to see if there is a change in perceived quality from one generation to the next, or simply if there is a difference between the samples warranting the desired range of effects of the effect unit.

In Picbreeder [14] neural networks are used to produce pictures which resembles human recognisable shapes. It is likewise the aim of this project to make the neural network evolve sound into human recognisable 'shapes' in the sense that the human ear can recognise one sound as pleasing and another as not.

A user survey (see section 5) is conducted to asses the success of the project along these lines. Comparing sound samples recorded at intervals will provide a way to measure if the CPPN-based system creates enough difference between samples to say something about the range of the system, and thereby judge the success of the project.

[9] http://chromium.googlecode.com/svn-history/r2003/trunk/samples/audio/specification/specification.html#MixerGainStructure-section

[10] http://www.amazon.com/Mixing-Engineers-Handbook-Bobby-Owsinski/dp/128542087X

[11] https://developer.mozilla.org/en-US/docs/Web/API/ScriptProcessorNode

3 Wave Theory

A periodic waveform can be expressed as $Y(t) = A * sin(2 * pi * f * t)$. Where A is the wave amplitude, f is the frequency, t is the time in seconds and Y is the displacement (in arbitrary units.)

The amplitude is measured in units according to occasion: movement of the eardrum listening to a 500 Hz sound-wave the amplitude will typically be measured in pressure units but one could measure the eardrums movement in direct units like millimetres or micrometres. For the output from a microphone, the amplitude will be measured in volts. The amplitude of sea waves would probably be measured directly in metres or feet.

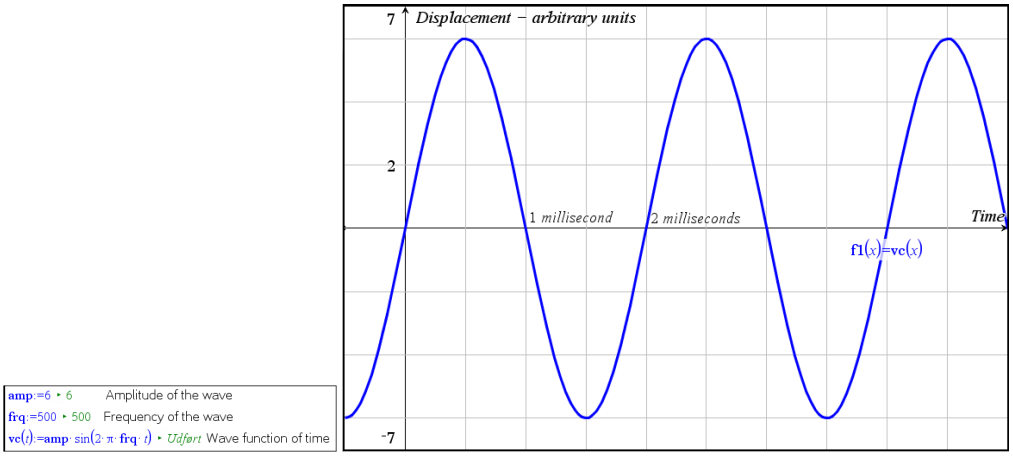


Figure 3.1: periodic wave

In figure 3.1 one full wave period takes exactly 2 milliseconds. Therefore the number of full cycles per second (i.e. the frequency) is $1s/0.002s/cycle = 500cycles/s = 500Hz$.

As another example figure 3.2 shows how two waves with a small difference in frequency produces sound pulses. This is also an audio effect used to fine-tune stringed instruments (and other instruments, e.g. the Moog synthesizer). As one string is playing the other can be tuned to it by continuously diminishing the pulsing between them by decreasing or increasing

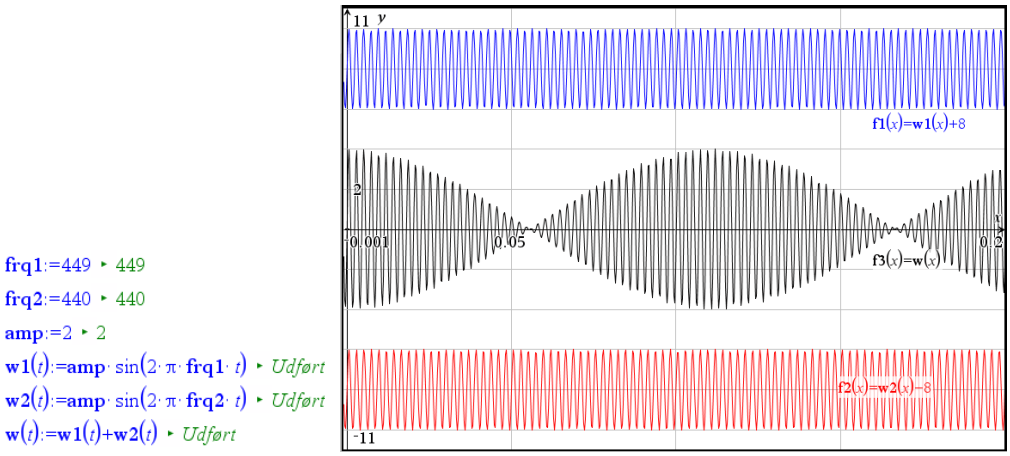


Figure 3.2: Wave pulses

the pitch until their pulse frequency are the same. At which point the two strings (or other) have the same frequency.

Since wave modulation is a matter of doing some arithmetic operation on the input samples it seems straight forward for this project to try out several modulation techniques/types. An obvious method of doing this would be to make modulation type a module of the system. e.g. by using a strategy pattern to enable plugging in new modulation forms at runtime. The final system provides two ways of doing modulation, simple amplitude modulation and linearly increasing the amplitude of an input buffer: $Math.sqrt(sample/buffersize)$. The latter has no discernible effect on the audio. A discussion on why can be found in the conclusion (see section 6.2).

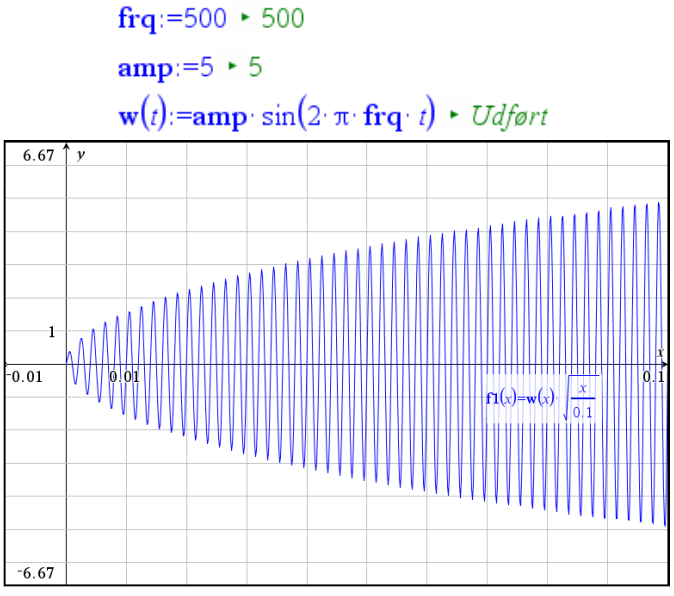


Figure 3.3: Modulating waves by the square root

Figure 3.3 shows an example of modulation by square root.

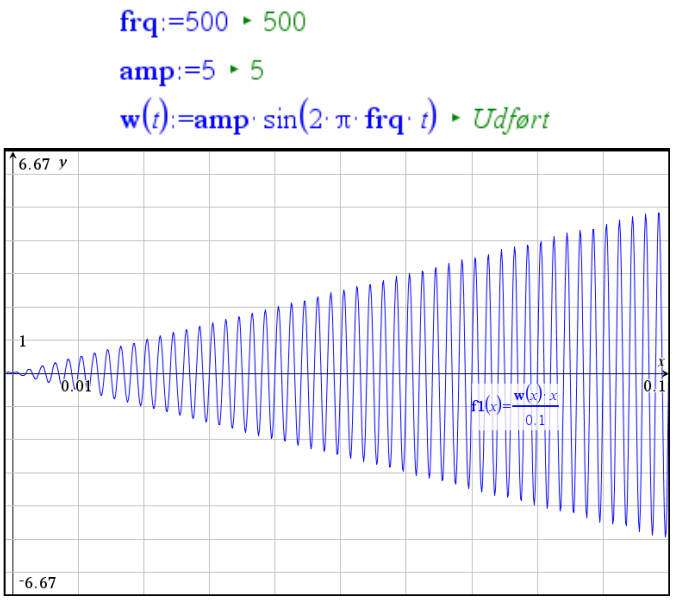


Figure 3.4: linearly growing wave amplitude

Figure 3.4 shows an example of linearly growing amplitude.

4 Technical Description

In which a few key concepts of the project are explained.

4.1 Technical Description

The system consists conceptually of three parts: CPPNs, processing audio, and a GUI.

4.1.1 CPPN Input and Output

Picture evolution projects based on CPPN-NEAT, such as Picbreeder, work with two-dimensional inputs to the CPPN networks representing the coordinates of pixels in the pictures [26], [22]. In contrast, the waveform timbre evolution in Breedesizer accepts only one-dimensional inputs representing the samples of the waveform to be generated. The input to one network is the output of the previous.

The numbers in a sequence that represents a single cycle of a periodic waveform falls within the range from -1 to 1. To ensure continuity between the ends of the waveforms the absolute of values y from that range is fed in as the input signal (comparable to the d coordinate [29]). To produce repetition in the outputs from the CPPNs, $sin(n * abs(y))$ is fed into another input node, where the integer value n is adjustable in the user interface, to allow different frequencies of repetition, (see figure 4.1). To observe the Repetition with Variation property of CPPNs [29], the user interface provides the option to avoid the absolute value function on the y values passed to the sine wave calculation, (see figure 4.2).

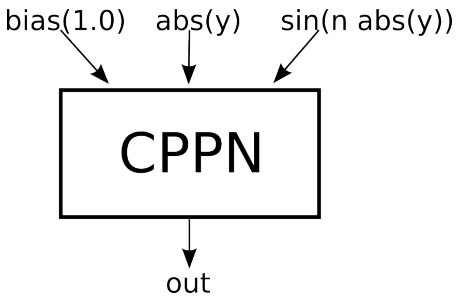


Figure 4.1: CPPN Inputs. Numbers in a sequence that represents a single cycle of a waveform, and periodic inputs as sine waves produced from multiples of the numbers in that same sequence [22].

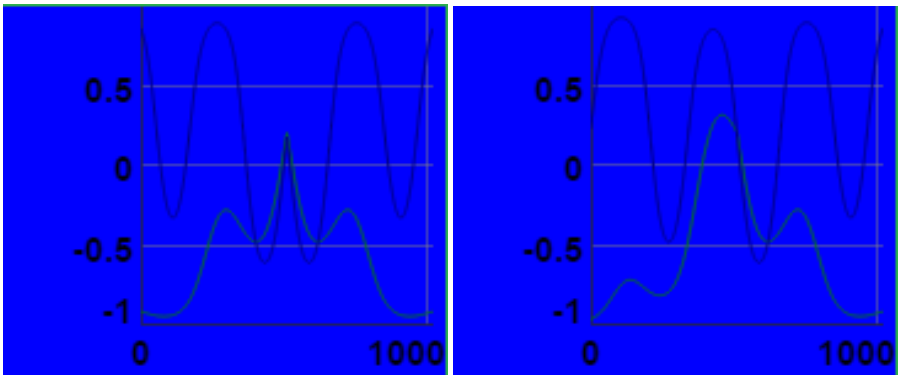


Figure 4.2: Waveforms output from the same CPPN, either produced from inputs restricted to absolute values (left), or with no restriction on the periodic input node (right)

The activation functions for each hidden neuron in the CPPNs are chosen from the set of Gaussian, Bipolar sigmoid, sine and linear, each with a .25 probability.

The user interface provides two sliders, one adjusting the probability of adding a connection and the other for adding a node. The user interface provides a slider for adjusting the frequency of the periodic wave input to each CPPN. The effect of this is not as dramatic as if the project had been exploring frequency modulation.

Sliders are provided in the user interface for adjusting how many times to attempt a mutation on each individual, which are set to five times for individuals in the initial population (providing variety in the waveforms in the initial population), and then defaulted to a setting of one attempt.

The waveforms output by the CPPNs are decomposed to their constituent frequencies by a Fourier transform. The result is a table of coefficients in a Fourier series representing a periodic waveform. Using the Web Audio API, those Fourier coefficients are provided as a parameter to the *createPeriodicWave* method which outputs a periodic wave, or a wavetable. This project uses those same wavetables but instead of frequency modulation the wavetables are used in the simpler process of modulating the amplitude of an input audio signal.

4.1.2 Processing

Via the web audio API a stream of user input is opened from a microphone input or, if present, from a USB input device. The audio stream format is raw PCM data. An audio stream is a buffer consisting of channels: one for mono, two for stereo, etc. The length of the buffer is the number of frames in it. Each frame contains a sample for each channel. All much in the spirit of the PCM format [1].

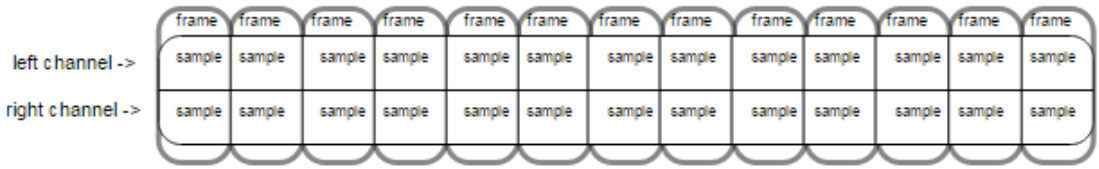


Figure 4.3: PCM raw data

The sample rate is the number of frames that will play in one second (measured in Hz.). e.g. a buffer at 41000Hz will be 41000 frames long, containing 41000 sampes (or frames) when mono, but 82000 frames when stereo, etc.

The project system loops over the buffer, manipulating samples. The processed samples are sent to the computer soundcard as raw PCM . Amplitude modulation [2] is achieved by multiplying the samples from the neural network wavetable onto the individual samples of the input audio.

[1] http://en.wikipedia.org/wiki/Pulse-code_modulation
[2] http://en.wikipedia.org/wiki/Pulse-code_modulation

```
// pseudo code
for (var channel = 0; channel < outputBuff.numberOfChannels; channel++) {
  var inputData = inputBuff.getChannelData(channel);
  var outputData = outputBuff.getChannelData(channel);
  for (var sample = 0; sample < inputBuff.length; sample++) {
    outputData[sample] *= cppnWave[sample];
  }
}
```

Figure 4.4: Processing individual PCM samples

4.2 User Experience

Audio effect units often have several adjustable parameters. This project has developed a unit with twenty-nine user adjustable parameters. A thorough walk-through of the audio effect unit's interface can be found in appendix A.

When the system loads in the browser, the user is presented with an effect unit resembling conventional mixers or audio units. The interface is split into sections, each representing a logical unit of functionality. User adjustable sliders has been made into rounded 'knobs' [3] to resemble the look and feel of a physical effect unit.

Different sections allow for adjusting parameters of the effects: CPPN, distortion, reverb, compression etc.

One section presents ten graphs (like in the Breedesizer project), each representing a CPPN waveshape. By clicking a graph, the input signal is modulated by that waveshape and sent to the output. The user can select which waveform(s) to use as the parent for the next generation of waveforms and then click 'evolve' to get the next generation of waveshapes. The user can select how many generations to evolve at a time. If more than one, the system uses the same parents for all.

The user can select between working with a default audio sample or a plugged in instrument/microphone. The default setting is the audio sample so when the page loads all the user has to do is select a waveform and she can immediately start working with the system.

The adjustable parameters have been connected as a real effect unit in a 'bus' structure. i.e., each effect has its own local adjustable parameters and each effect is at the same time connected to a 'send' which determines the overall amount of that effect which is routed to the main output (see section 2.1).

When the user is satisfied with the result he/she can record the resulting audio. The user can output the parameters of a chosen CPPN in JSON format. While recording, the system also saves the CPPN parameters in JSON format.

An example of a resulting CPPN network in the JSON format can be found in appendix B. The accompanying wav file can be found in the project directory (wav examples.) The sample is a generation 0 sound.

[3] https://github.com/aterrien/jQuery-Knob

5 User Survey

The aim of the project is to investigate the feasibility of neuro-evolutionary techniques in sound production. In particular, can CPPN produced effect units create a broader range of effects than ordinary effect units? To assess the results of the project a small user survey was conducted.

5.1 Method and Samples

Five samples were recorded, *sound1, sound2,..,sound5* during a single session with the system. The sound samples were created using the same basic sound sample for each recording. The basic sample contains no effect of any kind. The same settings of the parameters of the system was used throughout the recordings. i.e., no parameters were changed between recordings. Thus, the only difference between the recorded samples is the CPPN network. For each sample except the first one (generation 0 is created by the system with random input) two parents were chosen. Apart from generation 0, these generations were then chosen subjectively by a human, and a sample was recorded. The sounds were played to attendees in the survey and they were then asked to establish an order of sound recording. If the attendees can recognise a gradual change in the characters of the sounds, the CPPN-based sound system is functioning as intended.

The increasing complexity in the sounds created by the CPPN-based system is illustrated in figure 5.1 by the difference between two of the samples used in the survey: the first generation and the last. For another visual comparison of CPPN-based audio samples see the spectral analysis graphics in appendix C.

The sound samples used in the survey.

- sound2 (generation 0): http://www.itu.dk/people/msci/nikolai/xfct.wav,
- sound5 (generation 1): http://www.itu.dk/people/msci/nikolai/tcfx.wav,
- sound1 (generation 7): http://www.itu.dk/people/msci/nikolai/ctfx.wav,

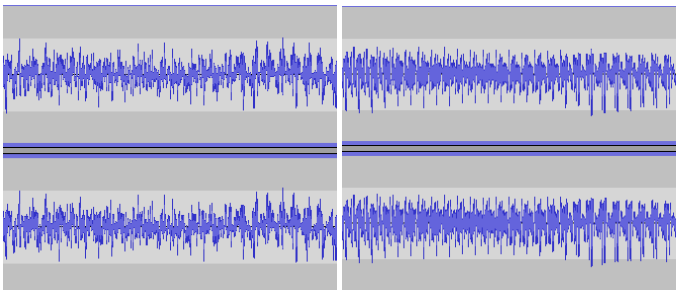


Figure 5.1: First generation (left) and last generation (right) waveshapes. Increasing complexity.

- sound4 (generation 8): http://www.itu.dk/people/msci/nikolai/txcf.wav,
- sound3 (generation 11): http://www.itu.dk/people/msci/nikolai/tfxc.wav

The reason for the jumps between generation numbers, as opposed to a one-step progression, is to achieve a noticeable change in sound quality from sample to sample. Some generations were simply used as in-betweeners before a good generation was recorded. This was deemed acceptable since the user is the objective function (see section 2.3.3) and will do exactly that.

The first two samples, generation 0 and 1, has a strong low-frequency trembling and almost dark quality to them. The subsequent three generations show a continuous decrease in the amount of 'darkness' and low-frequency. This is consistent with the choice of parents. The parents of the next generation were consistently picked on the basis of least darkness.

A scheme is used here to convert between the sample title and the actual generation (an unnecessary complication designed to hide any logical progression from the participants of the survey). Thus, sound2 is generation 0 (the first generation, created without parents by the CPPN network.) sound 5 is generation 1 (the first generation created by IEC.) sound 1 is generation 7 (recorded after seven IEC controlled evolutions). sound 4 is generation 8 and sound 3 is generation 11.

5.2 Questionnaire

Question 1: *"Order the 5 sounds.Which is the most crazy (give it the number 5), and which is least crazy (give it the number 1). Give each a unique number"*

A Likert-type scale (Likert-items) [1] was used for this question to score the sound samples along a range from 1 to 5.

The goal with this question is to see if the CPPNs are able to evolve the effects into a range of exiting new effects. If the attendees can discern an order of the samples then the CPPNs have been able to evolve the effect.

[1] http://en.wikipedia.org/wiki/Likert_scale

Question 2: *"Could you imagine hearing sounds like these on a recording or at a concert?"*

The Likert-type scale (Likert-items) was used for this question to score the entire system. The range here was: [*(not really)*; *(possibly)*; *(yes, definitely)*; *(don't know)*].

This question investigates how far the project is toward a real-life application. If the majority of answers are positive as to hearing the effects in real-life contexts then the CPPN generated effects are capable of producing relevant audio effects.

In two more questions the attendees were asked to compare the individual sounds. These questions are not used in the analysis, since an error in the questionnaire means attendees compared the wrong sounds. See section 5.4 for a discussion on this.

5.3 Survey Results

51 replies to the survey was attained. 58.82 % of the replies are given by males, 41.18 % by females. 62.75 % are between 10 and 20 years old. 21.57% are between 20 and 30 years old. 7.84 % are between 40 and 50 years old. 5.88% are between 60 and 70 years old. No 30-40 year old and no 50-60 year old participated. The remaining 1.96 % of the participants are between 70 and 80 years old.

Only 7.84 % of the participants are full-time musicians. 29.41 % plays music and the rest are not musicians. Likewise, only 1.96 % of the participants compose music professionally. 9.80% compose music occasionally.

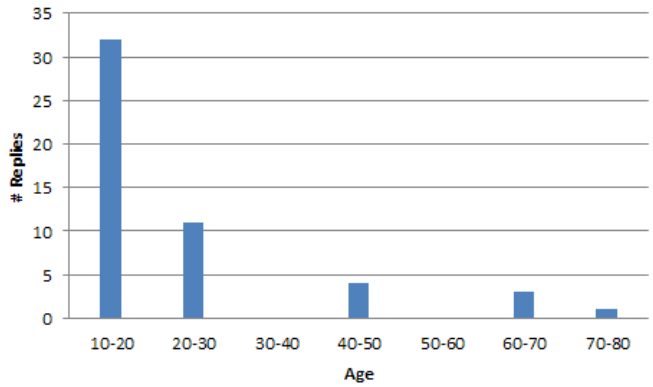


Figure 5.2: Age distribution

Figure 5.2 shows the age distribution in the survey. 62.7 % of participants are under 20 years of age.

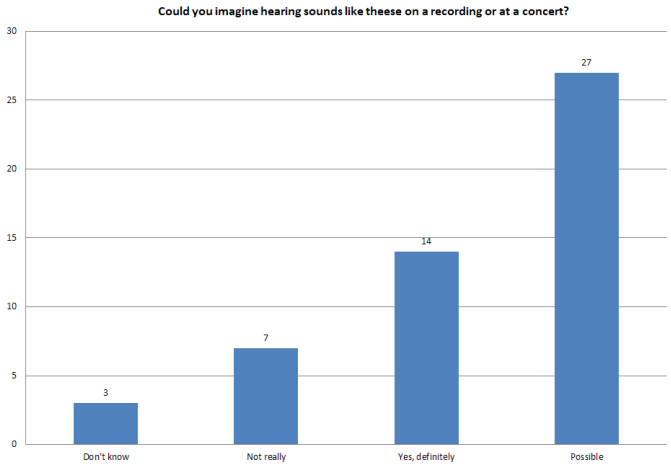


Figure 5.3: Applicability of CPPN audio effect units in other contexts

Figure 5.3: can the CPPN produced effects be presented in other contexts? The response points to the CPPN produced sound effect as a complementary to the standard audio effect unit. To some extend, this gives us a positive answer to one of our initial questions: can we use CPPNs to evolve new significantly different sound effects?

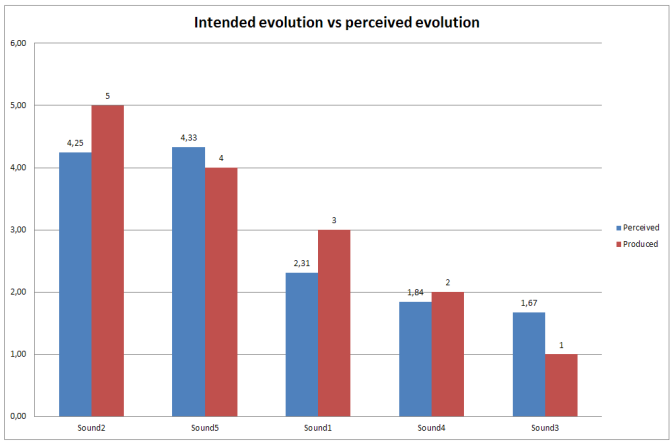


Figure 5.4: Juxtaposing surveyed order and sample order

Figure 5.4 depicts: in red, the sound samples in order of evolution, the youngest generations to the left, the oldest to the right. In blue, the surveyed order of evolution. Apart from the two first evolutions in the progression (called *sound2* and *sound5*) the columns proportionally match each other. This indicates a relationship between the evolved samples and the perceived evolution. The difference between the samples was picked up by the survey. The survey therefore is suggestive of CPPN-based systems' potential as a means of creating audio effect units of greater diversity than ordinary effect units.

5.3.1 Tests

To assess the results of the survey, two tests are conducted. A preliminary test, linear regression is performed to indicate the validity of the data. Next, the mean and StdDev are calculated.

The hypothesis is that the participants in the survey are capable of hearing the order in which the samples were recorded. i.e., the audial difference between the sound samples is sufficiently large that the participants can identify it, thus confirming that CPPN-based system are capable of producing a great variety of effects.

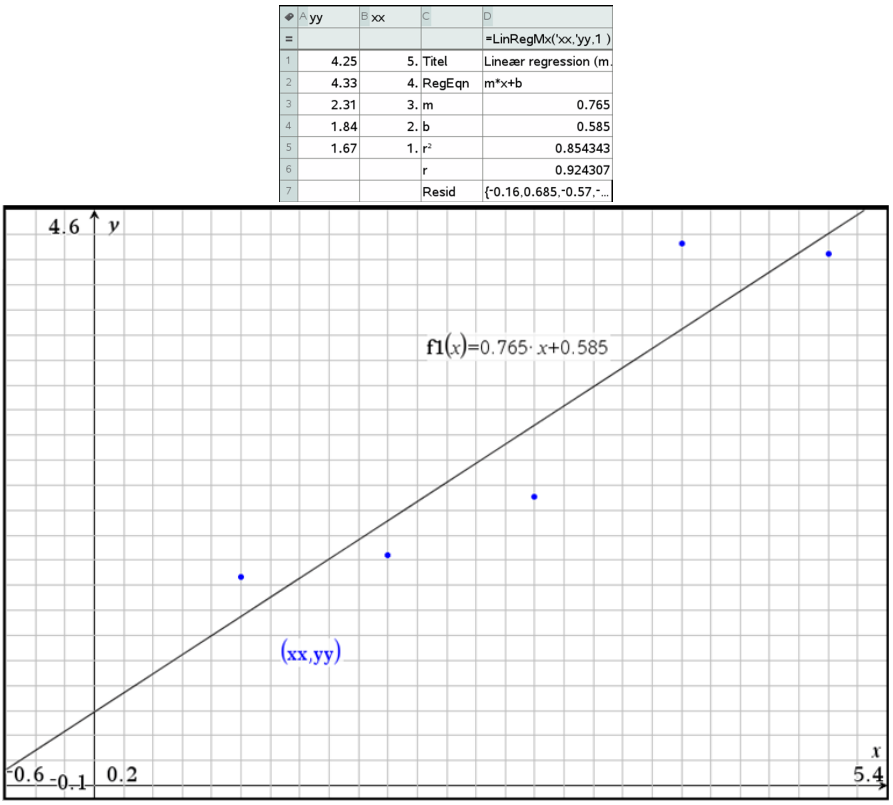


Figure 5.5: Linear regression

Figure 5.5: linear regression on the survey data. The independant variable is the audio sample ordering (1-5), the dependant variable is the observed order (the survey). The square of the correlation coefficient, r^2, has a value of 0.85. The relationship between the surveyed result and the expected outcome is less than 1, so a complete correlation between the surveyed and the known results can not be established.

Sound\Order	1	2	3	4	5	Participants
Sound 3	25	18	8	0	0	51
Sound 4	21	22	5	1	2	51
Sound 1	14	10	25	1	1	51
Sound 5	2	0	2	22	25	51
Sound 2	0	1	4	27	19	51
The table shows how the participants have chosen to order the sounds						
(14 participants have chosen to assign 1 to sound 1)						
(25 participants have chosen to assign 1 to sound 3)						

Figure 5.6: Surveyed sample order

Figure 5.6 shows how the participants ordered the sound samples. The table is ordered by the actual recording order. i.e., from youngest to oldest generation.

For each sound the Mean and standard deviation (StdDev) was then calculated.

Probability	1	2	3	4	5	Mean
Sound 3	0,2745	0,1961	0,4902	0,0196	0,0196	2,3
Sound 4	0,0000	0,0196	0,0784	0,5294	0,3725	4,3
Sound 1	0,4902	0,3529	0,1569	0,0000	0,0000	1,7
Sound 5	0,4118	0,4314	0,0980	0,0196	0,0392	1,8
Sound 2	0,0392	0,0000	0,0392	0,4314	0,4902	4,3

Probability	Varians	StdDev	Min	Max	Expected
Sound 3	0,9	0,9	1,4	3,2	1
Sound 4	1,3	1,1	3,1	5,4	2
Sound 1	0,5	0,7	0,9	2,4	3
Sound 5	0,9	0,9	0,9	2,8	4
Sound 2	1,5	1,2	3,1	5,6	5

Figure 5.7: the probability that a participant chooses a certain number for a given sample

Figure 5.7 The table shows the probability that a participant assigns a given number to a given sound. The table is ordered by the actual recording order. i.e., from youngest to oldest generation. The yellow coloured cell shows that the probability that a participant selects 2 for sound 2 is 1,96%. The blue coloured cell shows that the probability that a participant selects 1 for sound 4 is 49,02%. It can be seen in the table that for every sound sample the expected value falls within the Mean ± StdDev. The hypothesis can, on this basis, not be rejected. The participants *can* hear the difference between the sound samples.

As to the initial question: Can CPPN produced effect units create a broader range of effects than ordinary effect units? The study suggest the CPPN-based system developed is capable of providing a range of sound effects. The sound samples used in this survey was spread over only 11 generations of CPPN networks, yet participants were generally able to pick out the order of recordings. However, In the case of Sound2 (generation 0) and Sound5 (generation 1) the participants were not as certain. The fact that these were the two first generations might

explain why participants found it difficult to distinguish the character of the samples and their order. Being the first generations, these samples had very little guidance, so to speak. Still, the participants were generally certain that both samples belonged in the same end of the progression. If the spread over generations had been greater the audial difference between samples might have been clearer.

Another question posed in the introduction: can neuro-evolution make creating audio effects easier?, is not answered by this survey. See next section for an elaboration on this.

5.4 Afterthoughts

Great care must be taken in creating a survey and analysing data from it. The exact wording of the questions, the type of questionnaire etc. This survey was conducted under strict time constraints and some mistakes were made.

An attempt at obscuring to the participants any indication of progression in the recorded sound samples lead to more confusion than necessary. The five sound samples were given obscure names to cover their true order. This choice lead to trouble later in the survey. Several responses indicated trouble discerning the names of the samples. The collected answers and resulting analysis should take this into account. Furthermore, questions in the survey asking the attendees to compare individual sound samples failed to take into account the renaming. Those answers were therefore left out of the analysis.

One goal of the survey was to investigate the attitude of musicians toward the system. The lack of responses from musicians left this line of questioning irrelevant.

One question can not be answered by a user-survey: can neuro-evolution make creating audio effects easier? Such questions can only be answered by hands-on experience. Time did not allow for this.

A much more thorough user survey therefore is desirable for more decisive conclusions to be drawn. A survey involving musicians would be interesting as this would answer our last question: can neuro-evolution make creating guitar effects easier? It would also provide a means of testing the extend of the possibilities of the system.

6 Conclusion

6.1 Reflection

The developed system allows an endless variety of sound effects. CPPNs produce waveshapes until the user is satisfied with the result. Furthermore, each of those effects has 29 adjustable parameters. Each new generation produces sounds whose character inherits some traits from its parent(s) and has some unique variations. During development the oldest generation reached was 179. At around generation 80 (sometimes much earlier) the sounds start to gain an interesting 'metallic' high-pitch character, a distortion-like quality but only in the top frequencies. Who knows what characteristics generation 22478 will have? At the same time, though, chances of reaching a desired sound increase if the choice of parents is consistent from generation to generation, something very difficult to achieve.

The conducted user survey suggests evolutionary computation can be used in connection with audio production. The project was done as a proof of concept so much can be improved for it to have relevance in the music industry or in the social media music industry. Instead of working with amplitude, working with the frequency might produce interesting results. In this line of thought, the CPPN input could be changed into the input audio instead of wavetables. This is, perhaps, the most promising next step in developing a CPPN-based audio system like this one.

Latency is an issue in all real-time applications. At the start of this project high latency became an issue. The latency improved (decreased) after restructuring code. Optimizing the code even more might yield a further decrease in latency. The main suspect, when it comes to latency, though, is the web audio API's *scriptProcessorNode* and *audioBufferNode*. The system uses these to buffer input audio frames. Each audio-buffer contains 1024 samples which the system then modifies: each sample is multiplied by some number. It is likely this multiplication operation on a large buffer that is the culprit. For this reason, a big improvement to the system would be to try some kind of frequency modulation-based approach instead. This would put a lower demand on resources for a couple reasons: frequencies are made up of much less than 1024 units therefore not requiring as much processing, the web audio API has many tools for

working with the frequency of an audio signal.

A continuation of the project will be to get into the deeper matters of CPPN and neural networks in general. Given the opportunity there might be some easy tweaking of the parameters of the projects CPPN itself which could affect the audio produced in interesting ways.

6.2 Future Improvements

- The system needs general optimization and restructuring of the code. A thorough code optimization may in itself reduce latency. In link with this the system needs testing. It has been developed as a proof-of-concept and has not been under pressure from testing.
- By modifying the project to do FM instead of AM we may address two issues. One, the system gains speed and two, the pairing of CPPN on frequency instead of amplitude may produce more artistically appealing results.
- A logical improvement (or interesting experiment) would be to make the inputs to the CPPN networks be audio instead of the wavetables they use now. In combination with FM this might produce some very interesting sound effects.
- The project was done under strict time constraints. Therefore some concessions were made along the way, amongst which was the modulation part. The intention was to experiment with various modulation techniques but time was at a premium and the project ended up with amplitude modulation and modulation by the square root, which sounds very alike. A further investigation into modulation techniques may reveal improvements.
- Analysis of the project during development, done with Chrome DevTools [1], revealed memory leaks. Restructuring the code improved memory consumption. No conclusive solution came about before deadline.
- It takes little imagination to picture a commercial application of this kind of system. At least two such applications come to mind. One, a digital effect unit for recording studio purposes. Applications such as Protools [2] uses plug-in effects to modify sound so a evolutionary computation unit could easily fit in this scenario. Two, a hardware pedal using evolutionary computation might also be an viable application of the technology.

[1] https://developer.chrome.com/devtools
[2] http://www.avid.com/US/products/family/pro-tools

A Interface

When the system is first loaded, the browser asks permission to use the microphone. If the user accepts, the input to the system is routed via either the microphone or, if a USB device is plugged in, via it. If access is denied the system only works with loaded audio.

The user interface is split into sections in order to structure the many parameters logically. The first section contains a mute button and a spectrum analyser: a quick way to indicate to the user if any input audio is correctly set up.

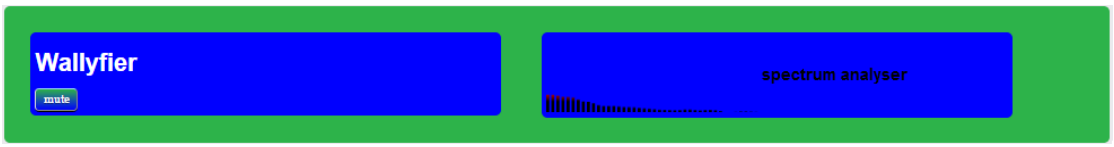


Figure A.1: Spectrum

The second section contains a checkbox that allows the user to shift between audio samples and live input (microphone or USB). If live is selected and an instrument is plugged into a USB device the user can play with the system in real-time (or, as near as the web audio api allows for.) If no USB device is available the microphone signal is used as the input.

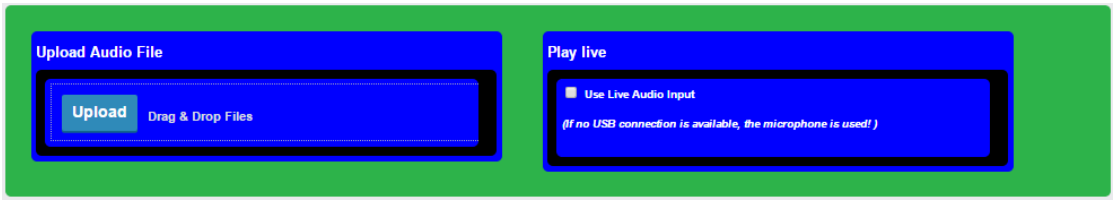


Figure A.2: Input choices

Section 3 contains global 'sends'. These determine how much of each of the effects are sent (with the input signal) to the final output. For total control the individual effect sections allow for fine-tuning the local parameters of the effect.

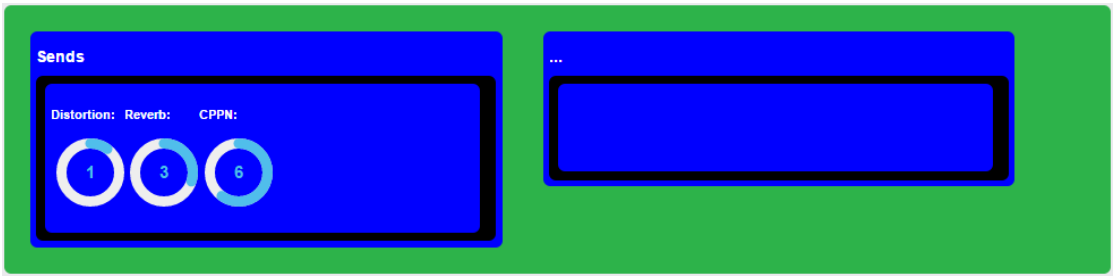


Figure A.3: Sends

Section 4 contains a standard compressor and a master gain. The user can adjust the compressors: 'Attack', the time it takes the compressor to activate, 'Ratio' is the factor determining how hard the compressor takes action, 'Release' is the time it takes the compressor to go back to the original signal and 'Threshold' is the volume at which the compressor kicks in. The area to the right holds a master volume, a mix between effect signal and clean signal (wet<->dry), a clip level which is the amount of amplitude cutting occurring in the modulated signal. It is used if the output 'crackles' too much. It lowers the amplitude of the modulated signal and affects a dampening on such crackles. Last in this section is a 'dry' amount -the recognized music business lingo for clean signal amount in the input to the modulation. Note wet-dry and Dry parameters are not functioning at the time of writing.

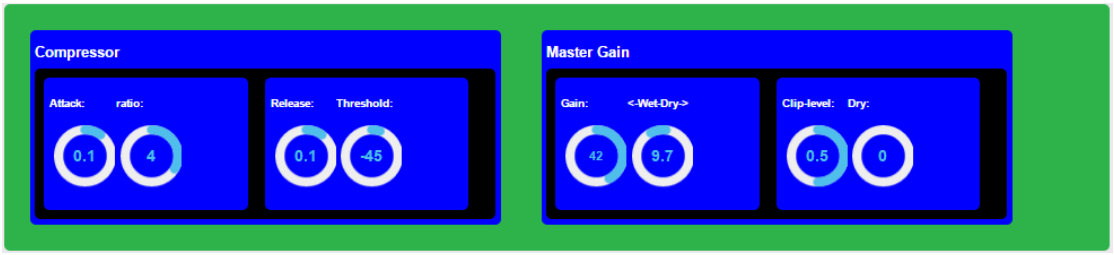


Figure A.4: Compressor and Master Gains

The next section contains parameters for affecting the evolutions of waveforms. some of the parameters stems from the Breedesizer project: Sliders for adjusting the probability of adding a connection and for adding a node. A slider for adjusting the frequency of the periodic wave input to each CPPN. Sliders for adjusting how many times to attempt a mutation on each individual

This project added a gain parameter affecting the overall volume of the modulated signal, a lowcut and gain parametric equalizer for cutting out the modulated signal at a given frequency and volume. On the right hand side of the section is a slider affecting the number of generations to evolve at a time (keeping the same parents for all of the evolutions if > 1), and modulation type checkboxes.

During code optimisation the equalizer and the 'Addition' modulation type stopped functioning properly and time did not allow for it to be corrected.

Figure A.5: CPPN and Modulation

The next section contains a simple reverb with parameters for duration and decay, as a signal gain and a reverse signal checkbox. To the right of the section, a simple distortion with parameters for distortion amount and distortion signal gain. Both effects are achieved by plain arithmetic on the audio buffer.

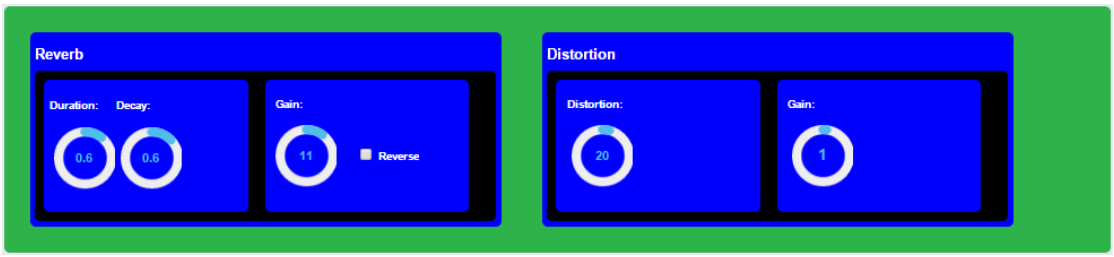


Figure A.6: Reverb and Distortion

The last section contains the CPPN waveforms. The user selects among these to hear the input signal modulated by a particular waveform after which, the user then selects which one(s) to use as parents for the next evolution.

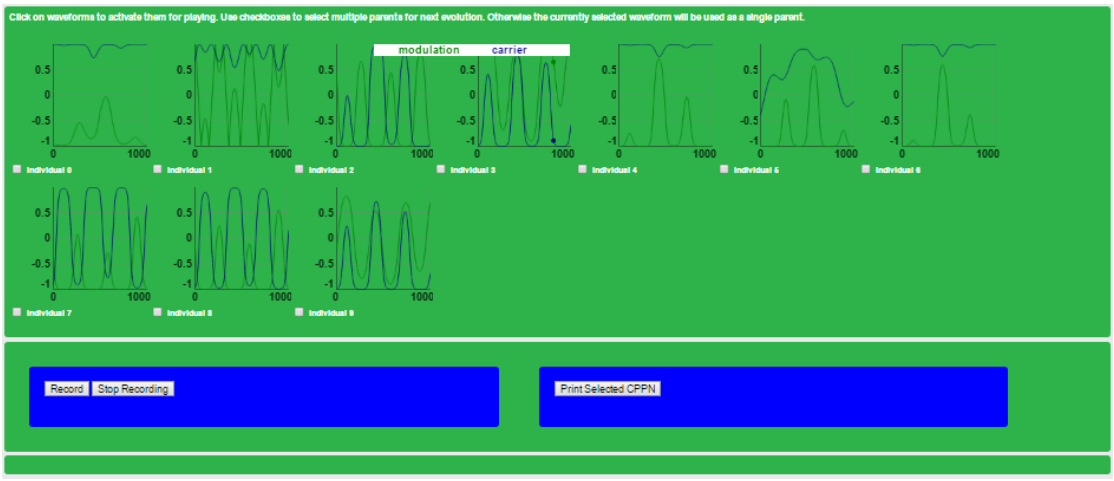


Figure A.7: The CPPN waveforms

B CPPN in JSON format


```
{
	"offspring": {
		"gid": 0,
		"fitness": 0,
		"nodes": [
			{
				"gid": "0",
				"activationFunction": "NullFn",
				"nodeType": "Bias",
				"layer": 0,
				"step": 0,
				"bias": 0
			},
			{
				"gid": "1",
				"activationFunction": "NullFn",
				"nodeType": "Input",
				"layer": 0,
				"step": 0,
				"bias": 0
			},
			{
				"gid": "2",
				"activationFunction": "NullFn",
				"nodeType": "Input",
				"layer": 0,
				"step": 0,
				"bias": 0
			},
			{
				"gid": "3",
				"activationFunction": "BipolarSigmoid",
				"nodeType": "Output",
```

```
                "layer": 10,
                "step": 0,
                "bias": 0
        },
        {
                "gid": "4",
                "activationFunction": "BipolarSigmoid",
                "nodeType": "Output",
                "layer": 10,
                "step": 0,
                "bias": 0
        },
        {
                "gid": "ci9v3z6s70013398j51quk6v978kn2yn4",
                "activationFunction": "spike",
                "nodeType": "Hidden",
                "layer": 5,
                "step": 0,
                "bias": 0
        }
],
"connections": [
        {
                "gid": "ci9v3z6s1000o398jcgcuifh38ho4lweu",
                "weight": -0.2235055584460497,
                "sourceID": "0",
                "targetID": "3",
                "a": 0,
                "b": 0,
                "c": 0,
                "d": 0,
                "modConnection": 0,
                "learningRate": 0,
                "isMutated": false
        },
        {
                "gid": "ci9v3z6s2000p398jyp0kdu87jbppoodh",
                "weight": -1.3789608449675144,
                "sourceID": "1",
                "targetID": "3",
                "a": 0,
                "b": 0,
                "c": 0,
                "d": 0,
                "modConnection": 0,
                "learningRate": 0,
                "isMutated": false
```

```
},
{
        "gid": "ci9v3z6s2000q398jp0yubx9m4t2rwm9j",
        "weight": -0.15430595772340894,
        "sourceID": "2",
        "targetID": "3",
        "a": 0,
        "b": 0,
        "c": 0,
        "d": 0,
        "modConnection": 0,
        "learningRate": 0,
        "isMutated": false
},
{
        "gid": "ci9v3z6s2000r398jknmfggvm4udctn24",
        "weight": 0.4785088310949504,
        "sourceID": "0",
        "targetID": "4",
        "a": 0,
        "b": 0,
        "c": 0,
        "d": 0,
        "modConnection": 0,
        "learningRate": 0,
        "isMutated": false
},
{
        "gid": "ci9v3z6s2000s398jk4qpmpx9y7idqtyt",
        "weight": 0.16706240139901637,
        "sourceID": "1",
        "targetID": "4",
        "a": 0,
        "b": 0,
        "c": 0,
        "d": 0,
        "modConnection": 0,
        "learningRate": 0,
        "isMutated": false
},
{
        "gid": "ci9v3z6s2000t398je4jtxk9cf28rxpgb",
        "weight": 0.05118288772646338,
        "sourceID": "2",
        "targetID": "4",
        "a": 0,
        "b": 0,
```

```
                "c": 0,
                "d": 0,
                "modConnection": 0,
                "learningRate": 0,
                "isMutated": false
        },
        {
                "gid": "ci9v3z6s70014398jhavnclx6va8h84lj",
                "weight": 1,
                "sourceID": "2",
                "targetID": "ci9v3z6s70013398j51quk6v978kn2yn4",
                "a": 0,
                "b": 0,
                "c": 0,
                "d": 0,
                "modConnection": 0,
                "learningRate": 0,
                "isMutated": false
        },
        {
                "gid": "ci9v3z6s70015398jurzl9cb7uv4bqjad",
                "weight": -0.15430595772340894,
                "sourceID": "ci9v3z6s70013398j51quk6v978kn2yn4",
                "targetID": "3",
                "a": 0,
                "b": 0,
                "c": 0,
                "d": 0,
                "modConnection": 0,
                "learningRate": 0,
                "isMutated": false
        }
],
"mutations": [],
"debug": false,
"behavior": {
        "behaviorList": null,
        "objectives": null
},
"realFitness": 0,
"age": 0,
"localObjectivesCompetition": [],
"meta": {},
"inputNodeCount": 2,
"inputAndBiasNodeCount": 3,
"outputNodeCount": 2,
"inputBiasOutputNodeCount": 5,
```

```
                "inputBiasOutputNodeCountMinus2": 3,
                "networkAdaptable": false,
                "networkModulatory": false,
                "connectionLookup": null,
                "nodeLookup": null,
                "parent": null
        },
        "parents": []
}
```

C Comparing Wave Spectrums

Comparing CPPNs by looking at the spectrals of different wav files.

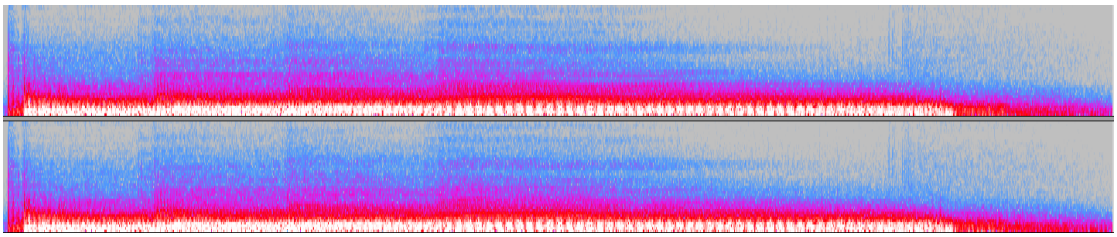


Figure C.1: A spectral analysis of generation 0

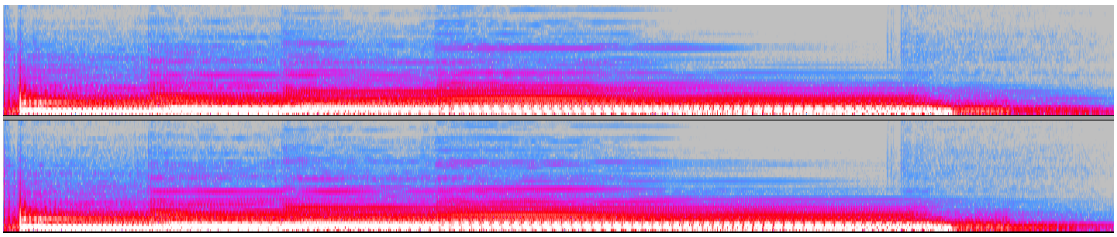


Figure C.2: A spectral analysis of generation 11

Bibliography

[1] Frequency modulation synthesis: Introduction to fm. URL http://rhordijk.home.xs4all.nl/G2Pages/FM.htm.

[2] Artificial neural networks. URL http://osp.mans.edu.eg/rehan/ann/Artificial%20Neural%20Networks.htm.

[3] Sound synthesis theory/oscillators and wavetables, ... URL http://en.wikibooks.org/wiki/Sound_Synthesis_Theory/Oscillators_and_Wavetables#Wavetables.

[4] The three sirens. URL http://www.the-three-sirens.info/binfo.html.

[5] Timbre (from wikipedia, the free encyclopedia), ... URL http://en.wikipedia.org/wiki/Timbre.

[6] Waveshaping. URL http://music.columbia.edu/cmc/musicandcomputers/chapter4/04_06.php.

[7] Reginald Bain. Waveform & timbre. URL http://in.music.sc.edu/fs/bain/atmi02/wt/index.html.

[8] John Biles, Peter Anderson, and Laura Loggi. Neural network fitness functions for a musical iga. 1996. URL http://scholarworks.rit.edu/other/184.

[9] Corban Brook. dsp.js. URL https://github.com/corbanbrook/dsp.js/.

[10] John Covert and David L Livingston. A vacuum-tube guitar amplifier model using a recurrent neural network. In *Southeastcon, 2013 Proceedings of IEEE*, pages 1–5. IEEE, 2013.

[11] Kenneth A. De Jong. *Evolutionary Computation a Unified Approach.* A Bradford Book, 2002.

[12] Miranda E. R. On the music of emergent behavior: What can evolutionary computation bring to the musician, 2003.

[13] Miranda E. R. At the crossroads of evolutionary computation and music: Self-programming synthesizers, swarm orchestras and the origins of melody. *Evolutionary Computation*, 12(2):137–158, 2004.

[14] Jeremiah T. Folsom-Kovarik. URL http://picbreeder.org.

[15] Michael Lee, Adrian Freed, and David Wessel. Neural networks for simultaneous classification and parameter estimation in musical instrument control, 1992. URL http://dx.doi.org/10.1117/12.139949.

[16] R.P Lippmann. An introduction to computing with neural nets. *ASSP Magazine*, 4(2): 4–22, 1987.

[17] James Michael McDermott. Evolutionary computation applied to the control of sound synthesis. 2008.

[18] Eduardo R Miranda, Larry Bull, François Gueguen, and Ivan S Uroukov. Computer music meets unconventional computing: towards sound synthesis with in vitro neuronal networks. *Computer Music Journal*, 33(1):9–18, 2009.

[19] Sébastien Molines. Using fourier transforms with the web audio api, 2014. URL http://www.sitepoint.com/using-fourier-transforms-web-audio-api/.

[20] Artemis Moroni. Vox populi: An interactive evolutionary system for algorithmic music composition, 2000.

[21] MICHAEL C. MOZER. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6 (2-3):247–280, 1994. doi: 10.1080/09540099408915726. URL http://dx.doi.org/10.1080/09540099408915726.

[22] Björn Þór Jónsson. Breedesizer. 2014. URL http://bthj.is/2015/01/02/breedesizer/.

[23] Gaëtan Renaudeau. Frequency modulation (fm) with web audio api, 2013. URL http://greweb.me/2013/08/FM-audio-api/.

[24] Sebastian Risi and Julian Togelius. Neuroevolution in games: State of the art and open challenges.

[25] Juan J Romero and Penousal Machado. *The art of artificial evolution: a handbook on evolutionary art and music*. Springer Science & Business Media, 2007.

[26] Jimmy Secretan, Nicholas Beato, David B D'Ambrosio, Adelein Rodriguez, Adam Campbell, Jeremiah T Folsom-Kovarik, and Kenneth O Stanley. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary Computation*, 19(3): 373–403, 2011.

[27] Beau Sievers. A young person's guide to the principles of music synthesis. URL http://beausievers.com/synth/synthbasics/.

[28] Kenneth Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[29] Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

[30] Hideyuki Takagi. Interactive evolutionary computation: System optimization based on human subjective evaluation. In *IEEE Int. Conf. on Intelligent Engineering Systems (INES'98)*, pages 17–19, 1998.

[31] IAN TAYLOR and MIKE GREENHOUGH. Modelling pitch perception with adaptive resonance theory artificial neural networks. *Connection Science*, 6(2-3):135–154, 1994. doi: 10.1080/09540099408915721. URL http://dx.doi.org/10.1080/09540099408915721.

[32] Nao Tokui and Hitoshi Iba. Music composition with interactive evolutionary computation. 17(2), 2000.

[33] Marques V. M. Interactive evolutionary computation in music. *Systems Man and Cybernetics (SMC)*, pages 3501–3507, 2010.

[34] Jianyi Ye and Shuangle Chen. Soundbreeder with multineat. URL http://web.cs.swarthmore.edu/~meeden/cs81/s14/papers/AndyLucas.pdf.

[35] ENGIN ZEKI. *DIGITAL MODELLING OF GUITAR AUDIO EFFECTS*. PhD thesis, MIDDLE EAST TECHNICAL UNIVERSITY, 2015.