SoundBreeder with MultiNEAT

Jianyi Ye and Shuangle Chen

Abstract

Evolutionary art, a technique developed by scientists to breed artistic works, began to draw increasingly more attention from both scientists and artists. This paper introduces the Soundbreeder program for evolving creative music and audio effect according to user preference. Soundbreeder has a user feedback mechanism after each evolution and allows the user to grade soundtracks; and the feedback from the user will be used for future evolution. Eventually, the network will learn the pattern that the user desires, and ideal music or sound effect will emerge. It takes a little effort in being consistent with previous choices, but it is a surprisingly powerful way to produce original sound patterns without the need for prior inspiration or musical talent.

1 Introduction

Evolutionary art allows art forms, pictures for example, to be bred almost like animals. On the one hand, with the development in adaptive robotics, more scientists are looking for practical applications for the neural network algorithms they've developed; on the other hand, more artists today are willing to trust and use computer as a very reliable and convenient tool in their working processes. One very typical product in creating evolutionary art is called the Picbreeder, developed by Jimmy Secretan and his development team. Picbreeder is a web application that allows the users to create and merge certain graphic patterns and evolve pictures based on personal preferences. With Picbreeder, "one can start with a butterfly and actually breed it into an airplane by selecting parents that look like airplanes. "

One good thing about such an evolutionary application is that it could really bring the idea of developing neural networks close to the mass public, and such application of developmental robotics is of great practical value. With proper development, evolutionary art can become a very useful tool for the general public. In the modern music industry, some artists are no longer satisfied with the limited forms of existing musical instruments. Rock bands begin to search for different possibilities in metal sound and other computer generated sounds. However, it usually takes long time for them to create the sound entirely by hand, so It would be nice to have such an evolutionary program that would bring about inspirations, or even develop cool sound effects for them.

Inspired by the Picbreeder, we decided to make a evolutionary art program that, ideally, will allow the users to breed sound patterns in a similar way that Picbreeders breeds pictures. Theoretically, this idea should work for soundtracks just as well as it works for pictures, because audio files and pictures have really similar file structures. The pictures consist of integer-valued pixels, and the numbers in each pixel determines its color. The total number of pixels within a fixed area determines the resolution of the picture. For audio files, we will be using the wav. file format in

our program, which will be introduced in later sections. Generally, the wav. file is a very simple and elegant audio file type. It consists of several parameters that determine its sound quality and total lasting time. The actual contents of the soundtrack, like pitches and volumes, are determined by the smallest component of a wav file, called frames. Frames are positive and negative integers, just like pixels in pictures.

The only big difference between these two file formats is the dimension. While the pictures are two-dimensional parallel values, the sound frames are one-dimensional consecutive values. It would be interesting to see how these two different types of media differ in their easiness in pattern finding and developing.

2 wav. File and External Libraries

2.1 wav. File

Wav. file is used as our primary audio source. Its full name is Waveform Audio File and is developed by IBM in the 1990s. We chose this format because it is arguably the most basic audio file and has little to no compression. Therefore they are easy to edit..

There are three key metrics of a war file: 1) whether it is mono or stereo, 2) number of frames in a file, which determines the length, and 3) the frame rate, which determines the speed.

Each wav file is comprised of numerous sound frames, and each frame is represented as an integer. The distance from the peak to zero decides the amplitude of the sound. The higher the peak, the louder the sound is. And the frequency decides the pitch of the sound. The higher the frequency, the higher the pitch is. Figure below is a visualization of a very simple wav file. As shown on the graph, the first section of sound is a very loud base sound; and the third section of sound has a much higher pitch yet is softer.

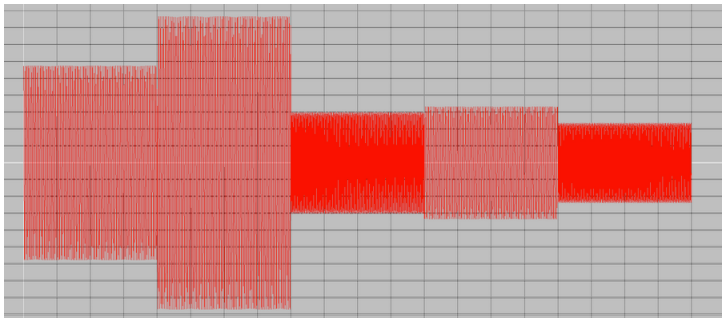

Figure 1: a typical wave file

2.2 Wave Library

The Wave library provides a convenient interface to the Wav. sound format. It allows us to load from an existing wav. file and to write a new wav. file. It does not support compression or decompression, but it does support mono/stereo. Although in our project, we are simplifying all of our wav. files to be mono. The essential functions in the Wave library that we used in our project are the followings:

`wav.open(file, mode)`

The open function allows users to either write in or read from a wav. file depending on the mode given by the user. The open function needs to be called prior to any other functions in this library.

`wav.setparams(tuple, file, mode)`

`*tuple is a struct consists of nchannels, sampwidth, framerate, nframes, comptype, and compname.`

The setparam function allows us to set several crucial features of the wav. file: the number of channels, the number of byte per audio sample, number of frames per second, total number of frames, compression type, and compression name. In our experiment, each wav. file has 10,000 frames and the frame rate is set to 2000 so all the sound tracks will last exactly 5 seconds. All the other parameters are set to their default values for simplicity of our experiment.

`wav.writeframes(data)`

The writeframes function allows users to put an integer into the wav. file as one single frame. These frames as a collection determines the pitch and volume.

2.3 Tkinter

Tkinter is Python's de-facto standard GUI (Graphical User Interface) package. It is a thin object-oriented layer on top of Tcl/Tk. It has a built-in a set of widgets including buttons, labels and so on. And programmers can easily add event listener to those widgets so that when a button is clicked, certain functions will be called.

2.4 MultiNEAT and HyperNEAT

MultiNEAT is a portable software library for performing neuroevolution. It is very similar to regular NEAT, also evolving neural networks through complexification, but it contains many external libraries that allows user to connect it with other algorithms quite easily. Just like NEAT, the neural networks in MultiNEAT begin evolution with very simple genomes which grow over multiple generations. The individuals in the evolving population are grouped into species by similarity, and each individual is only allow to compete within the same species.

HyperNEAT is an extension to NEAT by David D'Ambrosio, Jason Gauci and Kenneth Stanley, where the phenotypes are derived indirectly from special kind of neural networks known as Compositional Pattern Producing Networks (CPPNs). The genotypes are CPPNs evolved with NEAT, while the phenotypes are neural networks which have geometric structure. NEAT is used

to find the underlying geometric relationships of the problem.

Due to technical issues, we weren't able to use the HyperNEAT library directly, so we implemented one with the existing MultiNEAT and CPPN component writen by ourselves.

3 Hypothesis

Our hypothesis of the project is that the CPPN and NEAT will work for audio files like it does for pictures. That is, the CPPN would be able to produce sound tracks for different functions, and the NEAT would be able to develop proper neural networks that will produce outputs that make the CPPN produce sound tracks that are more and more in our favour. Regular patterns like symmetry and repetitions are expected.

At the first stage of our project, we were able to create simple soundtracks with one simple hardcoded instance of CPPN. with four different functions, tangent, sine, cosx /x, x*cosx. These four functions are chosen as our initial functions because they are all periodic functions and in our sound tracks we would like to see periodic patterns. Also, the functions we chose have very different patterns in the way they develop throughout the number line in the Euclidian space. When we first got the wav. files generated by our program, we extracted all the frames and put them into a picture. The sound patterns shown in our graphs make perfect sense according to the shape of those initial functions. The sine function gives relatively smooth sound patterns; the tangent has a more harsh sound because the frames go off the chart to infinitely large; the other two give the pattern of gradual increase or decrease of volumes and keys.

Now weve combined the CPPN with the MultiNEAT, we are able to create the previously randomly generated weights for our CPPN with our neural networks to change the tunes of our soundtrack. Our hypothesis at this point is that the neural network would notice the trend of the user input and gradually begin to create sound patterns that the user will more likely be satisfied with. That is, the patterns will eventually converge after generations of selections and evolutions.

Figure 2: User interface

4 User Interface

Our current GUI supports users to play 5 tracks for one generation and rate each of them on a scale from 1-5. "Start Breeding will start generating the next sound tracks based on the rating for current sound tracks. Record will show the history of ratings picked by the user. And graph will provide visualizations of wav files, which is another way for user to make a choice. Sometimes the graph can reveal interesting patterns and looking at a graph generally provides more measurable details than hearing a sound.

5 Algorithm

Our NEAT network (see Figure 3) is given the 2 inputs and will generate 7 outputs, which are scaled to become values only between 0 and 1. One of the inputs is the total number of distinct keys we wish to have in the soundtracks that are being created, and the other one is the number of volume changes we want to occur in the soundtracks we make.

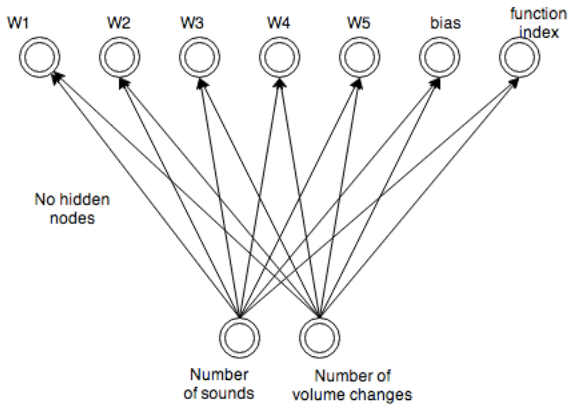

Figure 3: NEAT Network

Among the 7 outputs, 5 of them are used as weights for the CPPN network to produce input for the initial functions that creates frames for the soundtracks. One of them is bias. The last one is defined to be the function index value, which is also an number between 0 and 1. The value of the function index is used as a flag to determine which initial functions are to use in the production of one specific soundtrack.

In the special case when user chooses to have 5 pitches, 4 pivot points are put into the frame axis, they are frame 2000, frame 4000, frame 6000 and frame 8000. The distance between each point between each of those pivot points are calculated. Together with the current frame number, those five numbers are scaled first, then modified with the weights generated by the previous neural

networks. These five values are then used as inputs for the CPPN network (see Figure 4), which is set as having 5 inputs and 1 output, and the initial functions are determined by the function index value. Adding hidden node is set to have possibility of 0.1.

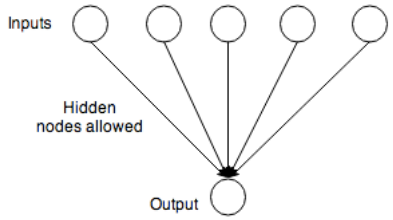

Figure 4: CPPN

Our CPPN is able to detect the four pivot points we set up and thus able to create distinct symmetricals and repeating patterns in respect of those four points, and this is how we managed to create different pitches within the same soundtrack.

For each generation, we will only have 5 individuals, after evaluation input by the users. A new generation will be created with the user inputs as fitness of the parents. We set the total generation number to 100, but we usually break the evolution in the middle when we found it not necessary to continue the evolution process any more.

6 Results

Due to the time limit, we are only able to evolve the sound patterns for around 400 generations. And since we don't have a very clear and static standard of preference, sometimes the patterns we create fail to become anything pleasing to human ears. However, we were once able to create an obvious converging pattern in a short 20 generations of evolutions.

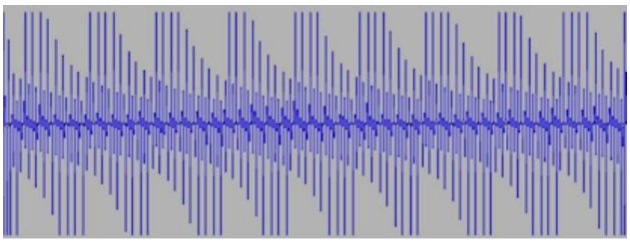

Figure 5: A patten from the tangent function.

At the very beginning of the evolutions, we were not too strict with our standards: we simply gave relatively high grades to smooth soundtracks, and gave very low grades to terrible ones, thus setting up the general trend of evolution early on. The neural network is relatively stable unless we gave very extreme grades. So in order to encourage the variety of the initial functions and sound patterns, we gave high grades to the soundtracks that used new functions.

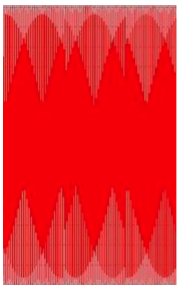

Figure 6: A patten from the sine function.

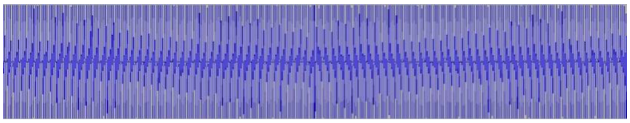

Figure 7: A symmetrical patten due to the pivot points.

As the experiments went on, there are some very interesting patterns we developed. Firstly, on the most local scale, there are sound patterns influenced by math functions like Tangent and Sine. On a larger scale, some generations have symmetry across the sound file, which is in line with the CPPNs characteristics in evolving pictures. In addition to those patterns exemplified in a single sound track, we all see trend and convergence across different generations.

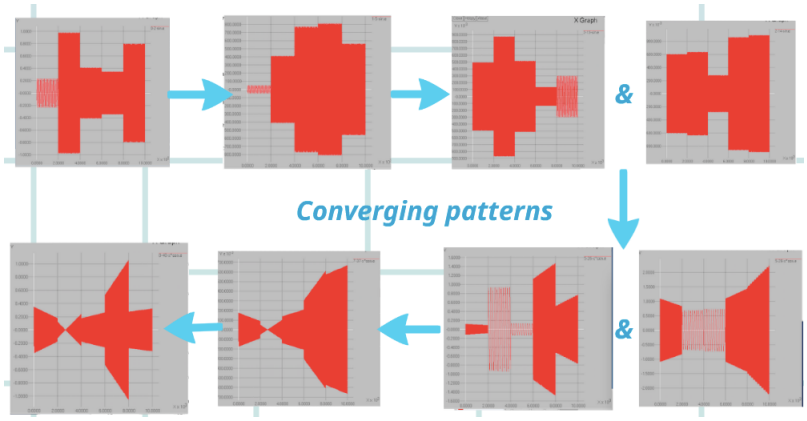

Figure 8: Converging patterns found in 20 generations of selections and evolutions.

Later on, we began to select patterns more meticulously. The grades we gave were generally lower than before. This slowed down the evolution process also made the it much more oriented. In the initial trial of evolution shown in Figure 8, we were looking for sound patterns with notable changes of volumes, and relatively smooth sound effects. Eventually, we got better and better results. When we extracted all the frames and put them into graphs very obvious converging pattern emerged as we expected,

Unfortunately, as previously mentioned, our program is still very inconsistent in its performance of forming converging patterns. Not all runs we had for our program worked out as well as the one previously shown.

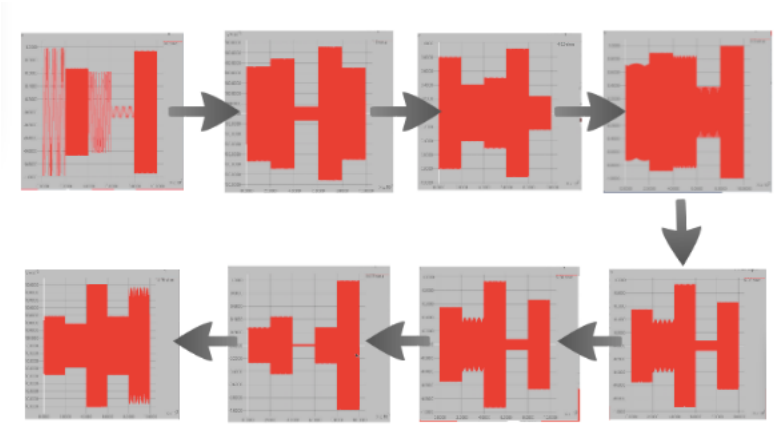

Figure 9: Patterns go back and forth without converging.

Figure 9 shows a typical example of a run that is not as satisfactory as the one shown in Figure 8. We can tell there are some similar general patterns found in the evolution, but the patterns are changing back and forth without settling down even after 50 generations of evolutions. We assume this is due to the unstable evaluation system we have chosen to use. That is, our standard of fitness is constantly varying, making it very confusing for the neural network to figure out what we want in limited generations.

7 Conlusions

In retrospect, SoundBreeder used MultiNEAT as its primary evolutionary algorithm and user preference as fitness function. It successfully generated interesting sound effect that has volume and pitch changes. From the graph, we can clearly see its usage of math functions and evolution of symmetry. As the evolution goes further, more and more converging cases emerge. The SoundBreeder has a potential for real application one day, with some further improvements:

1) Evolve Pivot Points.

Currently we have 4 fixed pivot points throughout our file. So the sounds we evolve will only have 5 tunes. In the future, we hope to evolve the number and position of pivot points. So the sounds will have more tunes throughout the file, and thus more twists and sound more natural.

2) Better Patterns.

Our current result is limited by the number of generations in our program. Since judgement of a human is needed for our fitness function, the evolution process will be very long and it is likely that we have to go through several hundred more generations to get more interesting patterns.

3) Frame Bounds.

Currently the program does not have limits on the pitch and amplitude during evolution. Therefore naturally some of our sounds are terrible for human ears. We would like to learn a bit more about music theory to find out the range of the pitch and amplitude, and set a bound on the frame.

8 Bibliography

[1] Compositional pattern producing networks: A novel abstraction of development, by Kenneth O. Stanley, in Genetic programming and evolvable machines, Vol. 8, (2007).

[2] Evolving neural network agents in the NERO video game, by Lee, Yosinski, Glette, Lipson, and Clune, in EvoApplications, (2013).

[3] Picbreeder: Collaborative Interactive Art Evolution (Genetic Art), Jimmy Secretan (Lead and Server Development), Last Modified: 05/09/2014

[4] "Introduction to wav. file format" , http://www.sonicspot.com/guide/wavefiles.html

[5] "wave library", https://docs.python.org/2/library/wave.html

[6] "Tkinter library", https://wiki.python.org/moin/TkInter

[7] "MultiNEAT library", http://multineat.com/download.html

[8] "modifying wav. files", http://stackoverflow.com/questions/19315919/extract-raw-audio-data-from-wav-files-using-libsndfile-in-c